ExTASY-0.2 Documentation

Release 0.2

RADICAL Group at Rutgers

February 26, 2016

Contents

Github Page

https://github.com/radical-cybertools/ExTASY

Mailing List

- Users : https://groups.google.com/forum/#!forum/extasy-project
- Developers : https://groups.google.com/forum/#!forum/extasy-devel

Build Status

Contents

CHAPTER 1

Introduction

1.1 What is ExTASY ?

ExTASY is a tool to run multiple Molecular Dynamics simulations which can be coupled to an Analysis stage. This forms a simulation-analysis loop which can be made to iterate multiple times. It uses a pilot framework, namely Radical Pilot to run a large number of these ensembles concurrently on most of the commonly used supercomputers. The complications of resource allocation, data management and task execution are performed using Radical Pilot and handled by the ExTASY.

ExTASY provides a command line interface, that along with specific configuration files, keeps the user's job minimal and free of the underlying execution methods and data management that is resource specific.

The coupled simulation-analysis execution pattern (aka ExTASY pattern) currently supports two usecases:

- **Gromacs** as the "Simulator" and **LSDMap** as the "Analyzer"
- **AMBER** as the "Simulator" and **CoCo** as the "Analyzer"

CHAPTER 2

Installation & Setup

2.1 Installation

This section describes the requirements and procedure to be followed to install the ExTASY package.

Note: Pre-requisites.The following are the minimal requirements to install the ExTASY module.

- python >= 2.7
- virtualenv >= 1.11
- pip >= 1.5
- Password-less ssh login to Stampede and/or Archer machine (help)

The easiest way to install ExTASY is to create virtualenv. This way, ExTASY and its dependencies can easily be installed in user-space without clashing with potentially incompatible system-wide packages.

Tip: If the virtualenv command is not available, try the following set of commands,

```
wget --no-check-certificate https://pypi.python.org/packages/source/v/virtualenv/virtualenv-1.11.tar
tar xzf virtualenv-1.11.tar.gz
python virtualenv-1.11/virtualenv.py --system-site-packages $HOME/ExTASY-tools/
source $HOME/ExTASY-tools/bin/activate
```

Step 1 : Create the virtualenv,

```
virtualenv $HOME/ExTASY-tools/
```

If your shell is BASH,

```
source $HOME/ExTASY-tools/bin/activate
```

If your shell is CSH,

Setuptools might not get installed with virtualenv and hence using pip would fail. Please look at https://pypi.python.org/pypi/setuptools for installation instructions.

```
source $HOME/ExTASY-tools/bin/activate.csh
```

To install the Ensemble MD Toolkit Python modules in the virtual environment, run:

```
pip install radical.ensemblemd
```

You can check the version of Ensemble MD Toolkit with the *ensemblemd-version* command-line tool.

```
0.3.14
```

Tip: If your shell is CSH you would need to do,

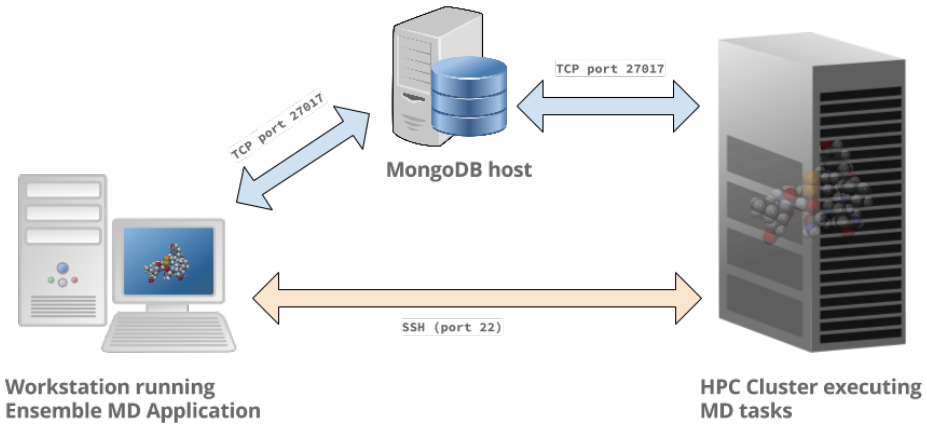
```
rehash
```

This will reset the PATH variable to also point to the packages which were just installed.

Installation is complete !

2.2 Preparing the Environment

ExTASY is developed using Ensemble MD Toolkit which is a client-side library and relies on a set of external software packages. One of these packages is radical.pilot, an HPC cluster resource access and management library. It can access HPC clusters remotely via SSH and GSISSH, but it requires (a) a MongoDB server and (b) a properly set-up SSH environment.



Note: For the purposes of the examples in this guide, we provide access to a mongodb url (mongodb://extasy:extasyproject@extasy-db.epcc.ed.ac.uk/radicalpilot). This is for trying out these examples **only** and is periodically purged. We recommend setting up your own mongodb instances for production simulations/experiments.

2.2.1 MongoDB Server

The MongoDB server is used to store and retrieve operational data during the execution of an Ensemble MD Toolkit application. The MongoDB server must be reachable on **port 27017** from **both**, the host that runs the Ensemble MD Toolkit application and the host that executes the MD tasks, i.e., the HPC cluster (see blue arrows in the figure above). In our experience, a small VM instance (e.g., Amazon AWS) works exceptionally well for this.

Warning: If you want to run your application on your laptop or private workstation, but run your MD tasks on a remote HPC cluster, installing MongoDB on your laptop or workstation won't work. Your laptop or workstations usually does not have a public IP address and is hidden behind a masked and firewalled home or office network. This means that the components running on the HPC cluster will not be able to access the MongoDB server.

A MongoDB server can support more than one user. In an environment where multiple users use Ensemble MD Toolkit applications, a single MongoDB server for all users / hosts is usually sufficient.

Install your own MongoDB

Once you have identified a host that can serve as the new home for MongoDB, installation is straight forward. You can either install the MongoDB server package that is provided by most Linux distributions, or follow the installation instructions on the MongoDB website:

http://docs.mongodb.org/manual/installation/

MongoDB-as-a-Service

There are multiple commercial providers of hosted MongoDB services, some of them offering free usage tiers. We have had some good experience with the following:

- https://mongolab.com/

2.2.2 HPC Cluster Access

In order to execute MD tasks on a remote HPC cluster, you need to set-up password-less SSH login for that host. This can either be achieved via an ssh-agent that stores your SSH key's password (e.g., default on OS X) or by setting up password-less SSH keys.

Password-less SSH with ssh-agent

An ssh-agent asks you for your key's password the first time you use it and then stores it for you so that you don't have to enter it again. On OS X (>= 10.5), an ssh-agent is running by default. On other Linux operating systems you might have to install or launch it manually.

You can test whether an ssh-agent is running by default on your system if you log-in via SSH into the remote host twice. The first time, the ssh-agent should ask you for a password, the second time, it shouldn't. You can use the `ssh-add` command to list all keys that are currently managed by your ssh-agent:

```
%> ssh-add -l
4096 c3:d6:4b:fb:ce:45:b7:f0:2e:05:b1:81:87:24:7f:3f /Users/enmdtk/.ssh/rsa_work (RSA)
```

For more information on this topic, please refer to this article:

- http://mah.everybody.org/docs/ssh

Password-less SSH keys

Warning: Using password-less SSH keys is really not encouraged. Some sites might even have a policy in place prohibiting the use of password-less SSH keys. Use ssh-agent if possible.

These instructions were taken from http://www.linuxproblem.org/art_9.html

Follow these instructions to create and set-up a public-private key pair that doesn't have a password.

As `user_a` on host `workstation`, generate a pair of keys. Do not enter a passphrase:

```
user_a@workstation:~> ssh-keygen -t rsa

Generating public/private rsa key pair.
Enter file in which to save the key (/home/a/.ssh/id_rsa):
Created directory '/home/a/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/a/.ssh/id_rsa.
Your public key has been saved in /home/a/.ssh/id_rsa.pub.
The key fingerprint is:
3e:4f:05:79:3a:9f:96:7c:3b:ad:e9:58:37:bc:37:e4 a@A
```

Now use ssh to create a directory ~/.ssh as `user_b` on `cluster`. (The directory may already exist, which is fine):

```
user_a@workstation:~> ssh user_b@cluster mkdir -p .ssh
user_b@cluster's password:
```

Finally append `usera_a`'s new public key to `user_b@cluster:.ssh/authorized_keys` and enter `user_b`'s password one last time:

```
user_a@workstation:~> cat .ssh/id_rsa.pub | ssh user_b@cluster 'cat >> .ssh/authorized_keys'
user_b@cluster's password:
```

From now on you can log into `cluster` as `user_b` from `workstation` as `user_a` without a password:

```
user_a@workstation:~> ssh user_b@cluster
```

Note: Depending on your version of SSH you might also have to do the following changes:

- Put the public key in `.ssh/authorized_keys2` (note the **2**)
- Change the permissions of .ssh to 700
- Change the permissions of .ssh/authorized_keys2 to 640

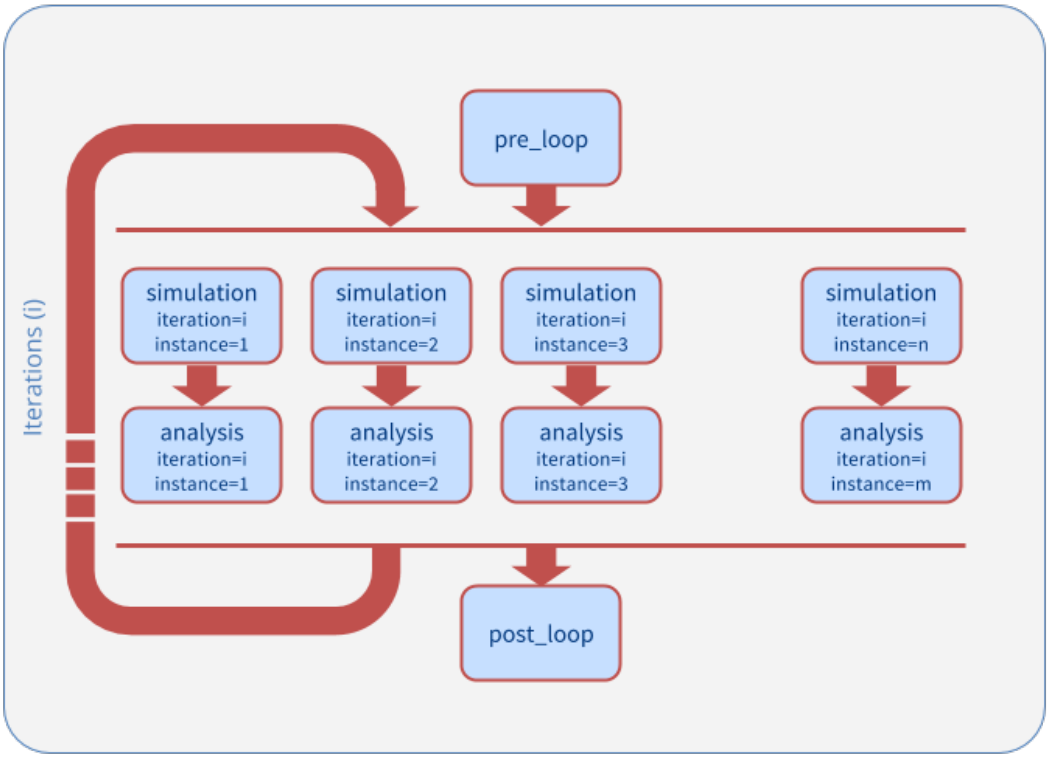
CHAPTER 3

Getting Started

ExTASY 0.2 uses the EnsembleMD Toolkit API for composing the application. In this section we will run you through the basics building blocks of the API. We will introduce the SimulationAnalysisLoop pattern and then work through simple examples using the same pattern in the next section. Once you are comfortable with these examples, we will then move to two molecular dynamics applications created using this API.

3.1 SimulationAnalysisLoop (SAL): The Pattern

The SAL pattern supports multiple iterations of two chained bag of tasks (BoT). The first bag consists of 'n' instances of simulations followed by the second bag which consists of 'm' instances of analysis. These analysis instances work on the output of the simulation instances and hence they are chained. There can be multiple iterations of these two BoTs. Depending on the application, it is possible to have the simulation instances of iteration 'i+1' work on the output of the analysis instances of iteration 'i'. There also exist two steps - pre_loop and post_loop to perform any pre- or post- processing. A graphical representation of the pattern is given below:



There are also a set of data references that can be used to reference the data in a particular step or instance.

- `$PRE_LOOP` - References the pre_loop step
- `$PREV_SIMULATION` - References the previous simulation step with the same instance number.
- `$PREV_SIMULATION_INSTANCE_Y` - References instance Y of the previous simulation step.
- `$SIMULATION_ITERATION_X_INSTANCE_Y` - References instance Y of the simulation step of iteration number X.
- `$PREV_ANALYSIS` - References the previous analysis step with the same instance number.
- `$PREV_ANALYSIS_INSTANCE_Y` - References instance Y of the previous analysis step.
- `$ANALYSIS_ITERATION_X_INSTANCE_Y` - References instance Y of the analysis step of iteration number X.

3.2 Components of the API

There are three components that the user interacts with in order to implement the application:

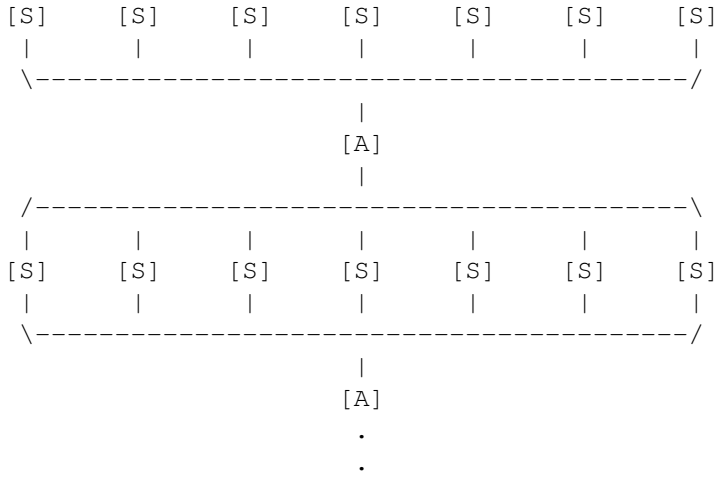
- **Execution Context**: The execution context can be see as a container that acquires the resources on the remote machine and provides application level control of these resources.
- **Execution Pattern**: A pattern can be seen as a parameterized template for an execution trajectory that implements a specific algorithm. A pattern provides placeholder methods for the individual steps or stages of an execution trajectory. These placeholders are populated with Kernels that get executed when it's the step's turn to be executed. In ExTASY, we will be using the SAL pattern.
- **Application Kernel**: A kernel is an object that abstracts a computational task in EnsembleMD. It represents an instantiation of a specific science tool along with its resource specific environment.

CHAPTER 4

Running Generic Examples

4.1 Multiple Simulations Single Analysis Application with SAL pattern

This example shows how to use the SAL pattern to execute 4 iterations of a simulation analysis loop with multiple simulation instances and a single analysis instance. We skip the `pre_loop` step in this example. Each `simulation_step` generates 16 new random ASCII files. One ASCII file in each of its instances. In the `analysis_step`, the ASCII files from each of the simulation instances are analyzed and character count is performed on each of the files using one analysis instance. The output is downloaded to the user machine.



Warning: In order to run this example, you need access to a MongoDB server and set the `RADICAL_PILOT_DBURL` in your environment accordingly. The format is `mongodb://hostname:port`.

4.1.1 Run locally

- Step 1: View the example source *below*. You can download the generic examples using the following:

```
curl -k -O  https://raw.githubusercontent.com/radical-cybertools/ExTASY/extasy_data/tarballs/generic
tar xfz generic.tar.gz
cd generic
```

Note: The files in the above link are configured to run for the CECAM workshop. The source at the end of this page is generic and might require changes.

- Step 2: Run the `multiple_simulations_single_analysis`:

```
python multiple_simulations_single_analysis.py
```

Once the script has finished running, you should see the character frequency files generated by the individual ensembles (cfreqs-1.dat) in the in the same directory you launched the script in. You should see as many such files as were the number of iterations. Each analysis stage generates the character frequency file for all the files generated in the simulation stage every iteration.

Note: Environment variable RADICAL_ENMD_VERBOSE is set to REPORT in the python script. This specifies the verbosity of the output. For more verbose output, you can use INFO or DEBUG.

4.1.2 Run remotely

By default, simulation and analysis steps run on one core your local machine:

```
SingleClusterEnvironment(
resource="local.localhost",
cores=1,
walltime=30,
username=None,
project=None
)
```

You can change the script to use a remote HPC cluster and increase the number of cores to see how this affects the runtime of the script as the individual simulations instances can run in parallel. For example, execution on xsede.stampede using 16 cores would require:

```
SingleClusterEnvironment(
resource="xsede.stampede",
cores=16,
walltime=30,    #minutes
username=None,  # add your username here
project=None # add your allocation or project id here if required
)
```

4.1.3 Example Script

Download `multiple_simulations_single_analysis.py`

```
#!/usr/bin/env python



__author__       = "Vivek <vivek.balasubramanian@rutgers.edu>"
__copyright__    = "Copyright 2014, http://radical.rutgers.edu"
__license__      = "MIT"
__example_name__ = "Multiple Simulations Instances, Single Analysis Instance Example (MSSA)"


```

```
import sys
import os
import json

from radical.ensemblemd import Kernel
from radical.ensemblemd import SimulationAnalysisLoop
from radical.ensemblemd import EnsemblemdError
from radical.ensemblemd import SingleClusterEnvironment


# ------------------------------------------------------------------------------
# Set default verbosity

if os.environ.get('RADICAL_ENMD_VERBOSE') == None:
    os.environ['RADICAL_ENMD_VERBOSE'] = 'REPORT'

# ------------------------------------------------------------------------------
#
class MSSA(SimulationAnalysisLoop):
    """MSMA exemplifies how the MSMA (Multiple-Simulations / Multiple-Analsysis)
       scheme can be implemented with the SimulationAnalysisLoop pattern.
    """
    def __init__(self, iterations, simulation_instances, analysis_instances):
        SimulationAnalysisLoop.__init__(self, iterations, simulation_instances, analysis_instances)


    def simulation_step(self, iteration, instance):
        """In the simulation step we
        """
        k = Kernel(name="misc.mkfile")
        k.arguments = ["--size=1000", "--filename=asciifile.dat"]
        return [k]

    def analysis_step(self, iteration, instance):
        """In the analysis step we use the ``$PREV_SIMULATION`` data reference
           to refer to the previous simulation. The same
           instance is picked implicitly, i.e., if this is instance 5, the
           previous simulation with instance 5 is referenced.
        """
        link_input_data = []
        for i in range(1, self.simlation_instances+1):
            link_input_data.append("$PREV_SIMULATION_INSTANCE_{instance}/asciifile.dat > asciifile-{i

        k = Kernel(name="misc.ccount")
        k.arguments            = ["--inputfile=asciifile-*.dat", "--outputfile=cfreqs.dat"]
        k.link_input_data      = link_input_data
        k.download_output_data = "cfreqs.dat > cfreqs-{iteration}.dat".format(iteration=iteration)
        return [k]


# ------------------------------------------------------------------------------
#
if __name__ == "__main__":

    try:

        # Create a new static execution context with one resource and a fixed
        # number of cores and runtime.
```

```
69          cluster = SingleClusterEnvironment(
70                          resource=resource,
71                          cores=1,
72                          walltime=15,
73                          #username=None,
74
75                          #project=None,
76                          #queue = None,
77
78                          #database_url=None,
79                          #database_name=None,
80          )
81
82          # Allocate the resources.
83          cluster.allocate()
84
85          # We set both the the simulation and the analysis step 'instances' to 16.
86          # If they
87          mssa = MSSA(iterations=4, simulation_instances=16, analysis_instances=1)
88
89          cluster.run(mssa)
90
91          cluster.deallocate()
92
93      except EnsemblemdError, er:
94
95          print "Ensemble MD Toolkit Error: {0}".format(str(er))
96          raise # Just raise the execption again to get the backtrace
```

In line 55, a SingleClusterEnvironment (Execution context) is created targetted to reserve **1** core on **localhost** for a duration of **30 mins**. In line 64, an allocation request is made for the particular execution context.

In line 16, we define the pattern class to be the SAL pattern. We skip the definition of the `pre_loop` step since we do not require it for this example. In line 27, we define the kernel that needs to be executed during the simulation step (mkfile) as well as the arguments to the kernel. In line 41, we define the kernel that needs to be executed during the analysis step (ccount). In lines 38-39, we create a list of references to the output data created in each of the simulation instances, in order to stage it in during the analysis instance (line 43).

In line 68, we create an instance of this MSSA class to run 4 iterations of 16 simulation instances and 1 analysis instance. We run this pattern in the execution context in line 70 and once completed we deallocate the acquired resources (line 72).

4.2 Multiple Simulations Multiple Analysis Application with SAL pattern

This example shows how to use the SAL pattern to execute 4 iterations of a simulation analysis loop with multiple simulation instances and multiple analysis instance. We skip the `pre_loop` step in this example. Each `simulation_step` generates 16 new random ASCII files. One ASCII file in each of its instances. In the `analysis_step`, the ASCII files from the simulation instances are analyzed and character count is performed. Each analysis instance uses the file generated by the corresponding simulation instance. This is possible since we use the same number of instances for simulation and analysis.The output is downloaded to the user machine.

```
[S]     [S]     [S]     [S]     [S]     [S]     [S]     [S]
 |       |       |       |       |       |       |       |
[A]     [A]     [A]     [A]     [A]     [A]     [A]     [A]
```

```
 |      |      |      |      |      |      |      |
[S]    [S]    [S]    [S]    [S]    [S]    [S]    [S]
 |      |      |      |      |      |      |      |
[A]    [A]    [A]    [A]    [A]    [A]    [A]    [A]
```

Warning: In order to run this example, you need access to a MongoDB server and set the `RADICAL_PILOT_DBURL` in your environment accordingly. The format is `mongodb://hostname:port`.

4.2.1 Run locally

- Step 1: View the example sources *below*. You can download the generic examples using the following (same link as above):

```
curl -k -O  https://raw.githubusercontent.com/radical-cybertools/ExTASY/extasy_data/tarballs/generic
tar xfz generic.tar.gz
cd generic
```

Note: The files in the above link are configured to run for the CECAM workshop. The source at the end of this page is generic and might require changes.

- Step 2: Run the `multiple_simulations_multiple_analysis`:

```
python multiple_simulations_multiple_analysis.py
```

Once the script has finished running, you should see the character frequency files generated by the individual ensembles (cfreqs-1-1.dat) in the in the same directory you launched the script in. You should see as many such files as were the number of iterations times the number of ensembles (i.e. simulation/analysis width). Each analysis stage generates the character frequency file for each of the files generated in the simulation stage every iteration.

Note: Environment variable RADICAL_ENMD_VERBOSE is set to REPORT in the python script. This specifies the verbosity of the output. For more verbose output, you can use INFO or DEBUG.

4.2.2 Run remotely

By default, simulation and analysis steps run on one core your local machine:

```
SingleClusterEnvironment(
resource="local.localhost",
cores=1,
walltime=30,
username=None,
project=None
)
```

You can change the script to use a remote HPC cluster and increase the number of cores to see how this affects the runtime of the script as the individual simulations instances can run in parallel. For example, execution on xsede.stampede using 16 cores would require:

```
SingleClusterEnvironment(
resource="xsede.stampede",
cores=16,
```

```
walltime=30,
username=None,  # add your username here
project=None # add your allocation or project id here if required
)
```

4.2.3 Example Script

Download multiple_simulations_multiple_analysis.py

```
#!/usr/bin/env python

__author__       = "Vivek <vivek.balasubramanian@rutgers.edu>"
__copyright__    = "Copyright 2014, http://radical.rutgers.edu"
__license__      = "MIT"
__example_name__ = "Multiple Simulations Instances, Multiple Analysis Instances Example (MSMA)"

import sys
import os
import json


from radical.ensemblemd import Kernel
from radical.ensemblemd import SimulationAnalysisLoop
from radical.ensemblemd import EnsemblemdError
from radical.ensemblemd import SingleClusterEnvironment


# ------------------------------------------------------------------------------
# Set default verbosity

if os.environ.get('RADICAL_ENMD_VERBOSE') == None:
    os.environ['RADICAL_ENMD_VERBOSE'] = 'REPORT'

# ------------------------------------------------------------------------------
#
class MSMA(SimulationAnalysisLoop):
    """MSMA exemplifies how the MSMA (Multiple-Simulations / Multiple-Analsysis)
       scheme can be implemented with the SimulationAnalysisLoop pattern.
    """
    def __init__(self, iterations, simulation_instances, analysis_instances):
        SimulationAnalysisLoop.__init__(self, iterations, simulation_instances, analysis_instances)


    def simulation_step(self, iteration, instance):
        """In the simulation step we
        """
        k = Kernel(name="misc.mkfile")
        k.arguments = ["--size=1000", "--filename=asciifile.dat"]
        return k

    def analysis_step(self, iteration, instance):
        """In the analysis step we use the ``$PREV_SIMULATION`` data reference
           to refer to the previous simulation. The same
           instance is picked implicitly, i.e., if this is instance 5, the
           previous simulation with instance 5 is referenced.
        """
        k = Kernel(name="misc.ccount")
```

```
49          k.arguments            = ["--inputfile=asciifile.dat", "--outputfile=cfreqs.dat"]
50          k.link_input_data      = "$PREV_SIMULATION/asciifile.dat".format(instance=instance)
51          k.download_output_data = "cfreqs.dat > cfreqs-{iteration}-{instance}.dat".format(instance=ins
52          return k
53
54
55  # ------------------------------------------------------------------------------
56  #
57  if __name__ == "__main__":
58
59      try:
60
61          # Create a new static execution context with one resource and a fixed
62          # number of cores and runtime.
63          cluster = SingleClusterEnvironment(
64                          resource=resource,
65                          cores=1,
66                          walltime=15,
67                          #username=None,
68                          #project=None,
69                          #queue = None,
70
71                          #database_url=None,
72                          #database_name=,
73          )
74
75          # Allocate the resources.
76          cluster.allocate()
77
78          # We set both the the simulation and the analysis step 'instances' to 8.
79          msma = MSMA(iterations=2, simulation_instances=8, analysis_instances=8)
80
81          cluster.run(msma)
82
83          cluster.deallocate()
84
85      except EnsemblemdError, er:
86
87          print "Ensemble MD Toolkit Error: {0}".format(str(er))
88          raise # Just raise the execption again to get the backtrace
```

In line 51, a SingleClusterEnvironment (Execution context) is created targetted to reserve **1** core on **localhost** for a duration of **30 mins**. In line 60, an allocation request is made for the particular execution context.

In line 16, we define the pattern class to be the SAL pattern. We skip the definition of the `pre_loop` step since we do not require it for this example. In line 27, we define the kernel that needs to be executed during the simulation step (mkfile) as well as the arguments to the kernel. In line 37, we define the kernel that needs to be executed during the analysis step (ccount). In lines 39, we refer to the output data created in the previous simulation step with the same instance number as that of the current analysis instance.

In line 63, we create an instance of this MSSA class to run 4 iterations of 16 simulation instances and 1 analysis instance. We run this pattern in the execution context in line 65 and once completed we deallocate the acquired resources (line 67).

CHAPTER 5

Running a Coco/Amber Workload

This section will discuss details about the execution phase. The input to the tool is given in terms of a resource configuration file and a workload configuration file. The execution is started based on the parameters set in these configuration files. In section 3.1, we discuss execution on Stampede and in section 3.2, we discuss execution on Archer.

5.1 Running on Stampede

5.1.1 Running using Example Workload Config and Resource Config

This section is to be done entirely on your **laptop**. The ExTASY tool expects two input files:

1. The resource configuration file sets the parameters of the HPC resource we want to run the workload on, in this case Stampede.
2. The workload configuration file defines the CoCo/Amber workload itself. The configuration file given in this example is strictly meant for the coco-amber usecase only.

Step 1: Create a new directory for the example,

```
mkdir $HOME/extasy-tutorial/
cd $HOME/extasy-tutorial/
```

Step 2: Download the config files and the input files directly using the following link.

```
curl -k -O  https://raw.githubusercontent.com/radical-cybertools/ExTASY/extasy_data/tarballs/coam-on
tar xvfz coam-on-stampede.tar.gz
cd coam-on-stampede
```

Step 3: In the coam-on-stampede folder, a resource configuration file `stampede.rcfg` exists. Details and modifications required are as follows:

Note: For the purposes of this example, you require to change only:

- UNAME
- ALLOCATION

The other parameters in the resource configuration are already set up to successfully execute the workload in this example.

Step 4: In the coam-on-stampede folder, a workload configuration file `cocoamber.wcfg` exists. Details and modifications required are as follows:

Note: All the parameters in the above example file are mandatory for amber-coco. There are no other parameters currently supported.

Step 5: You can find the executable script `'extasy_amber_coco.py'` in the coam-on-stampede folder.

Now you are can run the workload using :

```
python extasy_amber_coco.py --RPconfig stampede.rcfg --Kconfig cocoamber.wcfg
```

Note: Environment variable RADICAL_ENMD_VERBOSE is set to REPORT in the python script. This specifies the verbosity of the output. For more verbose output, you can use INFO or DEBUG.

Note: Time to completion: ~240 seconds (from the time job goes through LRMS)

5.2 Running on Archer

5.2.1 Running using Example Workload Config and Resource Config

This section is to be done entirely on your **laptop**. The ExTASY tool expects two input files:

1. The resource configuration file sets the parameters of the HPC resource we want to run the workload on, in this case Archer.
2. The workload configuration file defines the CoCo/Amber workload itself. The configuration file given in this example is strictly meant for the coco-amber usecase only.

Step 1: Create a new directory for the example,

```
mkdir $HOME/extasy-tutorial/
cd $HOME/extasy-tutorial/
```

Step 2: Download the config files and the input files directly using the following link.

```
curl -k -O  https://raw.githubusercontent.com/radical-cybertools/ExTASY/extasy_data/tarballs/coam-on-
tar xvfz coam-on-archer.tar.gz
cd coam-on-archer
```

Step 3: In the coam-on-archer folder, a resource configuration file `archer.rcfg` exists. Details and modifications required are as follows:

Note: For the purposes of this example, you require to change only:

- UNAME
- ALLOCATION

The other parameters in the resource configuration are already set up to successfully execute the workload in this example.

Step 4: In the coam-on-archer folder, a resource configuration file `cocoamber.wcfg` exists. Details and modifications required are as follows:

Note: All the parameters in the above example file are mandatory for amber-coco. There are no other parameters currently supported.

Step 5: You can find the executable script `'extasy_amber_coco.py'` in the coam-on-archer folder.

Now you are can run the workload using :

```
python extasy_amber_coco.py --RPconfig archer.rcfg --Kconfig cocoamber.wcfg
```

Note: Environment variable RADICAL_ENMD_VERBOSE is set to REPORT in the python script. This specifies the verbosity of the output. For more verbose output, you can use INFO or DEBUG.

Note: Time to completion: ~600 seconds (from the time job goes through LRMS)

5.3 Running on localhost

The above two sections describes execution on XSEDE.Stampede and EPSRC.Archer, assuming you have access to these machines. This section describes the changes required to the EXISTING scripts in order to get CoCo-Amber running on your local machines (label to be used = `local.localhost` as in the generic examples).

Step 1: You might have already guessed the first step. You need to create a SingleClusterEnvironment object targetting the localhost machine. You can either directly make changes to the `extasy_amber_coco.py` script or create a separate resource configuration file and provide it as an argument.

Step 2: The MD tools require some tool specific environment variables to be setup (AMBERHOME, PYTHONPATH, GCC, GROMACS_DIR, etc). Along with this, you would require to set the PATH environment variable to point to the binary file (if any) of the MD tool. Once you determine all the environment variables to be setup, set them on the terminal and test it by executing the MD command (possibly for a sample case). For example, if you have amber installed in $HOME as $HOME/amber14. You probably have to setup AMBERHOME to $HOME/amber14 and append $HOME/amber14/bin to PATH. Please check official documentation of the MD tool.

Step 3: There are three options to proceed.

- Once you tested the environment setup, next you need to add it to the particular kernel definition. You need to, first, locate the particular file to be modified. All the files related to EnsembleMD are located within the virtualenv (say "myenv"). Go into the following path: `myenv/lib/python-2.7/site-packages/radical/ensemblemd/kernel_plugins/md`. This path contains all the kernels used for the MD examples. You can open the amber.py file and add an entry for local.localhost (in `"machine_configs"`) as follows:

```
    ..
    ..
    "machine_configs":
    {

        ..
        ..

        "local.localhost":
```

```
        {
            "pre_exec"    : ["export AMBERHOME=$HOME/amber14", "export PATH=$HOME/amber14/bin:$PATH"
            "executable"  : ["sander"],
            "uses_mpi"    : False       # Could be True or False
        },

        ..
        ..

    }
    ..
    ..
```

This would have to be repeated for all the kernels.

- Another option is to perform the same above steps. But leave the "pre_exec" value as an empty list and set all the environment variables in your bashrc ($HOME/.bashrc). Remember that you would still need to set the executable as above.
- The third option is to create your own kernel plugin as part of your user script. These avoids the entire procedure of locating the existing kernel plugin files. This would also get you comfortable in using kernels other than the ones currently available as part of the package. Creating your own kernel plugins are discussed here

5.4 Understanding the Output of the Examples

In the local machine, a "output" folder is created and at the end of every checkpoint intervel (=nsave) an "iter*" folder is created which contains the necessary files to start the next iteration.

For example, in the case of CoCo-Amber on stampede, for 4 iterations with nsave=2:

```
coam-on-stampede$ ls
output/  cocoamber.wcfg  mdshort.in  min.in  penta.crd  penta.top  stampede.rcfg

coam-on-stampede/output$ ls
iter1/  iter3/
```

The "iter*" folder will not contain any of the initial files such as the topology file, minimization file, etc since they already exist on the local machine. In coco-amber, the "iter*" folder contains the NetCDF files required to start the next iteration and a logfile of the CoCo stage of the current iteration.

```
coam-on-stampede/output/iter1$ ls
1_coco.log    md_0_11.ncdf  md_0_14.ncdf  md_0_2.ncdf  md_0_5.ncdf  md_0_8.ncdf  md_1_10.ncdf  md_1_1
md_0_0.ncdf   md_0_12.ncdf  md_0_15.ncdf  md_0_3.ncdf  md_0_6.ncdf  md_0_9.ncdf  md_1_11.ncdf  md_1_1
md_0_10.ncdf  md_0_13.ncdf  md_0_1.ncdf   md_0_4.ncdf  md_0_7.ncdf  md_1_0.ncdf  md_1_12.ncdf  md_1_1
```

It is important to note that since, in coco-amber, all the NetCDF files of previous and current iterations are transferred at each checkpoint, it might be useful to have longer checkpoint intervals. Since smaller intervals would lead to heavy data transfer of redundant data.

On the remote machine, inside the pilot-* folder you can find a folder called "unit.00000". This location is used to exchange/link/move intermediate data. The shared data is kept in "unit.00000/" and the iteration specific inputs/outputs can be found in their specific folders (="unit.00000/iter*").

```
$ cd unit.00000/
$ ls
iter0/  iter1/  iter2/  iter3/  mdshort.in  min.in  penta.crd  penta.top  postexec.py
```

CHAPTER 6

Running a Gromacs/LSDMap Workload

This section will discuss details about the execution phase. The input to the tool is given in terms of a resource configuration file and a workload configuration file. The execution is started based on the parameters set in these configuration files. In section 4.1, we discuss execution on Stampede and in section 4.2, we discuss execution on Archer.

6.1 Running on Stampede

6.1.1 Running using Example Workload Config and Resource Config

This section is to be done entirely on your **laptop**. The ExTASY tool expects two input files:

1. The resource configuration file sets the parameters of the HPC resource we want to run the workload on, in this case Stampede.
2. The workload configuration file defines the GROMACS/LSDMap workload itself. The configuration file given in this example is strictly meant for the gromacs-lsdmap usecase only.

Step 1: Create a new directory for the example,

```
mkdir $HOME/extasy-tutorial/
cd $HOME/extasy-tutorial/
```

Step 2: Download the config files and the input files directly using the following link.

```
curl -k -O  https://raw.githubusercontent.com/radical-cybertools/ExTASY/extasy_data/tarballs/grlsd-o
tar xvfz grlsd-on-stampede.tar.gz
cd grlsd-on-stampede
```

Step 3: In the grlsd-on-stampede folder, a resource configuration file `stampede.rcfg` exists. Details and modifications required are as follows:

Note: For the purposes of this example, you require to change only:

- UNAME
- ALLOCATION

The other parameters in the resource configuration are already set up to successfully execute the workload in this example.

Step 4: In the grlsd-on-stampede folder, a workload configuration file `gromacslsdmap.wcfg` exists. Details and modifications are as follows:

Note: All the parameters in the above example file are mandatory for gromacs-lsdmap. If *ndxfile*, *grompp_options*, *mdrun_options* and *itp_file_loc* are not required, they should be set to None; but they still have to mentioned in the configuration file. There are no other parameters currently supported for these examples.

Step 5: You can find the executable script `'extasy_gromacs_lsdmap.py'` in the grlsd-on-stampede folder.

Now you are can run the workload using :

```
python extasy_gromacs_lsdmap.py --RPconfig stampede.rcfg --Kconfig gromacslsdmap.wcfg
```

Note: Environment variable RADICAL_ENMD_VERBOSE is set to REPORT in the python script. This specifies the verbosity of the output. For more verbose output, you can use INFO or DEBUG.

Note: Time to completion: ~13 mins (from the time job goes through LRMS)

6.2 Running on Archer

6.2.1 Running using Example Workload Config and Resource Config

This section is to be done entirely on your **laptop**. The ExTASY tool expects two input files:

1. The resource configuration file sets the parameters of the HPC resource we want to run the workload on, in this case Archer.
2. The workload configuration file defines the CoCo/Amber workload itself. The configuration file given in this example is strictly meant for the gromacs-lsdmap usecase only.

Step 1: Create a new directory for the example,

```
mkdir $HOME/extasy-tutorial/
cd $HOME/extasy-tutorial/
```

Step 2: Download the config files and the input files directly using the following link.

```
curl -k -O  https://raw.githubusercontent.com/radical-cybertools/ExTASY/extasy_data/tarballs/grlsd-on
tar xvfz grlsd-on-archer.tar.gz
cd grlsd-on-archer
```

Step 3: In the grlsd-on-archer folder, a resource configuration file `archer.rcfg` exists. Details and modifications required are as follows:

Note: For the purposes of this example, you require to change only:

- UNAME
- ALLOCATION

The other parameters in the resource configuration are already set up to successfully execute the workload in this example.

Step 4: In the grlsd-on-archer folder, a workload configuration file `gromacslsdmap.wcfg` exists. Details and modifications required are as follows:

Note: All the parameters in the above example file are mandatory for gromacs-lsdmap. If *ndxfile*, *grompp_options*, *mdrun_options* and *itp_file_loc* are not required, they should be set to None; but they still have to mentioned in the configuration file. There are no other parameters currently supported.

Step 5: You can find the executable script `'extasy_gromacs_lsdmap.py'` in the grlsd-on-archer folder.

Now you are can run the workload using :

```
python extasy_gromacs_lsdmap.py --RPconfig archer.rcfg --Kconfig gromacslsdmap.wcfg
```

Note: Environment variable RADICAL_ENMD_VERBOSE is set to REPORT in the python script. This specifies the verbosity of the output. For more verbose output, you can use INFO or DEBUG.

Note: Time to completion: ~15 mins (from the time job goes through LRMS)

6.3 Running on localhost

The above two sections describes execution on XSEDE.Stampede and EPSRC.Archer, assuming you have access to these machines. This section describes the changes required to the EXISTING scripts in order to get Gromacs-LSDMap running on your local machines (label to be used = `local.localhost` as in the generic examples).

Step 1: You might have already guessed the first step. You need to create a SingleClusterEnvironment object targetting the localhost machine. You can either directly make changes to the `extasy_gromacs_lsdmap.py` script or create a separate resource configuration file and provide it as an argument.

Step 2: The MD tools require some tool specific environment variables to be setup (`AMBERHOME`, `PYTHONPATH`, `GCC`, `GROMACS_DIR`, etc). Along with this, you would require to set the `PATH` environment variable to point to the binary file (if any) of the MD tool. Once you determine all the environment variables to be setup, set them on the terminal and test it by executing the MD command (possibly for a sample case). For example, if you have gromacs installed in `$HOME` as `$HOME/gromacs_5`. You probably have to setup `GROMACS_DIR` to `$HOME/gromacs-5` and append `$HOME/gromacs-5/bin` to `PATH`. Please check official documentation of the MD tool.

Step 3: There are three options to proceed.

- Once you tested the environment setup, next you need to add it to the particular kernel definition. You need to, first, locate the particular file to be modified. All the files related to EnsembleMD are located within the virtualenv (say "myenv"). Go into the following path: `myenv/lib/python-2.7/site-packages/radical/ensemblemd/kernel_plugins/md`. This path contains all the kernels used for the MD examples. You can open the `gromacs.py` file and add an entry for local.localhost (in `"machine_configs"`) as follows:

```
    ..
    ..
    "machine_configs":
    {

        ..
        ..
```

```
        "local.localhost":
        {
            "pre_exec"    : ["export GROMACS_DIR=$HOME/gromacs-5", "export PATH=$HOME/gromacs-5/bin:
            "executable"  : ["mdrun"],
            "uses_mpi"    : False       # Could be True or False
        },

        ..
        ..

    }
    ..
    ..
```

This would have to be repeated for all the kernels.

- Another option is to perform the same above steps. But leave the `"pre_exec"` value as an empty list and set all the environment variables in your bashrc (`$HOME/.bashrc`). Remember that you would still need to set the executable as above.
- The third option is to create your own kernel plugin as part of your user script. These avoids the entire procedure of locating the existing kernel plugin files. This would also get you comfortable in using kernels other than the ones currently available as part of the package. Creating your own kernel plugins are discussed here

6.4 Understanding the Output of the Examples

In the local machine, a "output" folder is created and at the end of every checkpoint intervel (=nsave) an "iter*" folder is created which contains the necessary files to start the next iteration.

For example, in the case of gromacs-lsdmap on stampede, for 4 iterations with nsave=2:

```
grlsd-on-stampede$ ls
output/  config.ini  gromacslsdmap.wcfg  grompp.mdp  input.gro  stampede.rcfg  topol.top

grlsd-on-stampede/output$ ls
iter1/  iter3/
```

The "iter*" folder will not contain any of the initial files such as the topology file, minimization file, etc since they already exist on the local machine. In gromacs-lsdmap, the "iter*" folder contains the coordinate file and weight file required in the next iteration. It also contains a logfile about the lsdmap stage of the current iteration.

```
grlsd-on-stampede/output/iter1$ ls
2_input.gro  lsdmap.log  weight.w
```

On the remote machine, inside the pilot-* folder you can find a folder called "unit.00000". This location is used to exchange/link/move intermediate data. The shared data is kept in "unit.00000/" and the iteration specific inputs/outputs can be found in their specific folders (="unit.00000/iter*").

```
$ cd unit.00000/
$ ls
config.ini  gro.py   input.gro   iter1/  iter3/    post_analyze.py  reweighting.py   run.py     split
grompp.mdp  gro.pyc  iter0/      iter2/  lsdm.py   pre_analyze.py   run_analyzer.sh  select.py  topol
```

CHAPTER 7

API Reference

7.1 Execution Context API

7.1.1 class radical.ensemblemd.SingleClusterEnvironment (resource, cores, walltime, database_url, queue = None, username = None, allocation = None, cleanup = False)

A static execution context provides a fixed set of computational resources.

- **name**: Returns the name of the execution context
- **allocate**(): Allocates the resources
- **deallocate**(): Deallocates the resources
- **run**(**pattern, force_plugin = None**): Create a new SingleClusterEnvironment instance
- **get_name**(): Returns the name of the execution pattern

7.2 Execution Pattern API

7.2.1 pre_loop()

The `radical.ensemblemd.Kernel` returned by pre_loop is executed before the main simulation-analysis loop is started. It can be used for example to set up structures, initialize experimental environments, one-time data transfers and so on.

***Returns*:** Implementations of this method must return either a single or a list of `radical.ensemblemd.Kernel` object(s). An exception is thrown otherwise.

7.2.2 simulation_step(iteration, instance)

The `radical.ensemblemd.Kernel` returned by simulation_step is executed once per loop iteration before analysis_step.

***Arguments*:**

- iteration [int] - The iteration parameter is a positive integer and references the current iteration of the simulation-analysis loop.

- instance [int] - The instance parameter is a positive integer and references the instance of the simulation step, which is in the range [1 .. simulation_instances].

Returns: Implementations of this method must return either a single or a list of `radical.ensemblemd.Kernel` object(s). An exception is thrown otherwise.

7.2.3 analysis_step(iteration, instance)

The `radical.ensemblemd.Kernel` returned by analysis_step is executed once per loop iteration after simulation_step.

Arguments:

- iteration [int] - The iteration parameter is a positive integer and references the current iteration of the simulation-analysis loop.
- instance [int] - The instance parameter is a positive integer and references the instance of the simulation step, which is in the range [1 .. simulation_instances].

Returns: Implementations of this method must return either a single or a list of `radical.ensemblemd.Kernel` object(s). An exception is thrown otherwise.

7.2.4 post_loop()

The `radical.ensemblemd.Kernel` returned by post_loop is executed after the main simulation-analysis loop has finished. It can be used for example to set up structures, initialize experimental environments and so on.

Returns: Implementations of this method must return a single `radical.ensemblemd.Kernel` object. An exception is thrown otherwise.

7.3 Application Kernel API

7.3.1 class radical.ensemblemd.Kernel (name, args = None)

The Kernel provides functions to support file movement as required by the pattern.

- **cores**: number of cores the kernel is using.
- **upload_input_data**: Instructs the application to upload one or more files or directories from the host the script is running on into the kernel's execution directory.

Example:

```
k = Kernel(name="misc.ccount")
k.arguments = ["--inputfile=input.txt", "--outputfile=output.txt"]
k.upload_input_data = ["/location/on/HOST/RUNNING/THE/SCRIPT/data.txt > input.txt"]
```

- **download_input_data**: Instructs the kernel to download one or more files or directories from a remote HTTP server into the kernel's execution directory.

Example:

```
k = Kernel(name="misc.ccount")
k.arguments = ["--inputfile=input.txt", "--outputfile=output.txt"]
k.download_input_data = ["http://REMOTE.WEBSERVER/location/data.txt > input.txt"]
```

- **copy_input_data**: Instructs the kernel to copy one or more files or directories from the execution host's filesystem into the kernel's execution directory.

Example:

```
k = Kernel(name="misc.ccount")
k.arguments = ["--inputfile=input.txt", "--outputfile=output.txt"]
k.copy_input_data = ["/location/on/EXECUTION/HOST/data.txt > input.txt"]
```

- **link_input_data**: Instructs the kernel to create a link to one or more files or directories on the execution host's filesystem in the kernel's execution directory.

Example:

```
k = Kernel(name="misc.ccount")
k.arguments = ["--inputfile=input.txt", "--outputfile=output.txt"]
k.link_input_data = ["/location/on/EXECUTION/HOST/data.txt > input.txt"]
```

- **download_output_data**: Instructs the application to download one or more files or directories from the kernel's execution directory back to the host the script is running on.

Example:

```
k = Kernel(name="misc.ccount")
k.arguments = ["--inputfile=input.txt", "--outputfile=output.txt"]
k.download_output_data = ["output.txt > output-run-1.txt"]
```

- **copy_output_data**: Instructs the application to download one or more files or directories from the kernel's execution directory to a directory on the execution host's filesystem.

Example:

```
k = Kernel(name="misc.ccount")
k.arguments = ["--inputfile=input.txt", "--outputfile=output.txt"]
k.download_output_data = ["output.txt > /location/on/EXECUTION/HOST/output.txt"]
```

- **get_raw_args()**: Returns the arguments passed to the kernel.
- **get arg(name)**: Returns the value of the kernel argument given by 'arg_name'.

7.4 Exceptions & Errors

This module defines and implement all ensemblemd Exceptions.

- **exception radical.ensemblemd.exceptions.EnsemblemdError(msg): EnsemblemdError is the base exception thrown by th**

```
Bases: exceptions.Exception
```

- **exception radical.ensemblemd.exceptions.NotImplementedError(method_name, class_name): NotImplementedError is th**

```
Bases: radical.ensemblemd.exceptions.EnsemblemdError
```

- **exception radical.ensemblemd.exceptions.TypeError(expected_type, actual_type): TypeError is thrown if a parameter of**

```
Bases: radical.ensemblemd.exceptions.EnsemblemdError
```

- **exception radical.ensemblemd.exceptions.FileError(message): FileError is thrown if something goes wrong related to file**

```
Bases: radical.ensemblemd.exceptions.EnsemblemdError
```

- **exception radical.ensemblemd.exceptions.ArgumentError(kernel_name, message, valid_arguments_set): A BadArgument**

```
Bases: radical.ensemblemd.exceptions.EnsemblemdError
```

- **exception radical.ensemblemd.exceptions.NoKernelPluginError(kernel_name): NoKernelPluginError is thrown if no kern**

```
Bases: radical.ensemblemd.exceptions.EnsemblemdError
```

- **exception radical.ensemblemd.exceptions.NoKernelConfigurationError(kernel_name, resource_key): NoKernelConfigura**

```
Bases: radical.ensemblemd.exceptions.EnsemblemdError
```

- **exception radical.ensemblemd.exceptions.NoExecutionPluginError(pattern_name, context_name, plugin_name): NoExec**

```
Bases: radical.ensemblemd.exceptions.EnsemblemdError
```

CHAPTER 8

Customization

8.1 Writing New Application Kernels

While the current set of available application kernels might provide a good set of tools to start, sooner or later you will probably want to use a tool for which no application Kernel exsits. This section describes how you can add your custom kernels.

We have two files, user_script.py which contains the user application which **uses** our custom kernel, new_kernel.py which contains the definition of the custom kernel. You can download them from the following links:

- `user_script.py`
- `new_kernel.py`

Let's first take a look at `new_kernel.py`.

```
from radical.ensemblemd.kernel_plugins.kernel_base import KernelBase

# ------------------------------------------------------------------------------
#
_KERNEL_INFO = {
    "name":         "sleep",        #Mandatory
    "description":  "sleeping kernel",        #Optional
    "arguments":   {                #Mandatory
        "--interval=": {
            "mandatory": True,        #Mandatory argument? True or False
            "description": "Number of seconds to do nothing."
                },
        },
    "machine_configs":             #Use a dictionary with keys as resource names and values specific
        {
            "local.localhost":
            {
                "environment" : None,         #list or None, can be used to set environment variables
                "pre_exec"    : None,             #list or None, can be used to load modules
                "executable"  : ["/bin/sleep"],         #specify the executable to be used
                "uses_mpi"    : False             #mpi-enabled? True or False
            },
        }
}

# ------------------------------------------------------------------------------
#
class MyUserDefinedKernel(KernelBase):
```

```
29
30      def __init__(self):
31
32          super(MyUserDefinedKernel, self).__init__(_KERNEL_INFO)
33              """Le constructor."""
34
35      # ------------------------------------------------------------------------------
36      #
37      @staticmethod
38      def get_name():
39          return _KERNEL_INFO["name"]
40
41      def _bind_to_resource(self, resource_key):
42          """This function binds the Kernel to a specific resource defined in
43          "resource_key".
44          """
45          arguments  = ['{0}'.format(self.get_arg("--interval="))]
46
47          self._executable  = _KERNEL_INFO["machine_configs"][resource_key]["executable"]
48          self._arguments   = arguments
49          self._environment = _KERNEL_INFO["machine_configs"][resource_key]["environment"]
50          self._uses_mpi    = _KERNEL_INFO["machine_configs"][resource_key]["uses_mpi"]
51          self._pre_exec    = _KERNEL_INFO["machine_configs"][resource_key]["pre_exec"]
52
53  # ------------------------------------------------------------------------------
```

Lines 5-24 contain information about the kernel to be defined. "name" and "arguments" keys are mandatory. The "arguments" key needs to specify the arguments the kernel expects. You can specify whether the individual arguments are mandatory or not. "machine_configs" is not mandatory, but creating a dictionary with resource names (same as defined in the SingleClusterEnvironment) as keys and values which are resource specific lets use the same kernel to be used on different machines.

In lines 28-50, we define a user defined class (of "KernelBase" type) with 3 mandatory functions. First the constructor, self-explanatory. Second, a static method that is used by EnsembleMD to differentiate kernels. Third, `_bind_to_resource` which is the function that (as the name suggests) binds the kernel with its resource specific values, during execution. In lines 41, 43-45, you can see how the "machine_configs" dictionary approach is helping us solve the tool-level heterogeneity across resources. There might be other ways to do this (if conditions,etc), but we feel this could be quite convenient.

Now, let's take a look at `user_script.py`

```
1   from radical.ensemblemd import Kernel
2   from radical.ensemblemd import Pipeline
3   from radical.ensemblemd import EnsemblemdError
4   from radical.ensemblemd import SingleClusterEnvironment
5
6   #Used to register user defined kernels
7   from radical.ensemblemd.engine import get_engine
8
9   #Import our new kernel
10  from new_kernel import MyUserDefinedKernel
11
12  # Register the user-defined kernel with Ensemble MD Toolkit.
13  get_engine().add_kernel_plugin(MyUserDefinedKernel)
14
15
16  #Now carry on with your application as usual !
17  class Sleep(Pipeline):
18
```

```
19      def __init__(self, instances,steps):
20          Pipeline.__init__(self, instances,steps)
21
22      def step_1(self, instance):
23          """This step sleeps for 60 seconds."""
24
25          k = Kernel(name="sleep")
26          k.arguments = ["--interval=10"]
27          return k
28
29
30  # ------------------------------------------------------------------------------
31  #
32  if __name__ == "__main__":
33
34      try:
35          # Create a new static execution context with one resource and a fixed
36          # number of cores and runtime.
37          cluster = SingleClusterEnvironment(
38                  resource="local.localhost",
39                  cores=1,
40                  walltime=15,
41                     username=None,
42                      project=None
43                  )
44
45          # Allocate the resources.
46          cluster.allocate()
47
48          # Set the 'instances' of the pipeline to 16. This means that 16 instances
49          # of each pipeline step are executed.
50          #
51          # Execution of the 16 pipeline instances can happen concurrently or
52          # sequentially, depending on the resources (cores) available in the
53          # SingleClusterEnvironment.
54          sleep = Sleep(steps=1,instances=16)
55
56          cluster.run(sleep)
57
58          cluster.deallocate()
59
60      except EnsemblemdError, er:
61
62          print "Ensemble MD Toolkit Error: {0}".format(str(er))
63          raise # Just raise the execption again to get the backtrace
```

There are 3 important lines in this script. In line 7, we import the get_engine function in order to register our new kernel. In line 10, we import our new kernel and in line 13, we register our kernel. **THAT'S IT**. We can continue with the application as in the previous examples.

8.2 Writing a Custom Resource Configuration File

A number of resources are already supported by RADICAL-Pilot, they are list in *List of Pre-Configured Resources*. If you want to use RADICAL-Pilot with a resource that is not in any of the provided configuration files, you can write your own, and drop it in $HOME/.radical/pilot/configs/<your_site>.json.

Note: Be advised that you may need system admin level knowledge for the target cluster to do so. Also, while RADICAL-Pilot can handle very different types of systems and batch system, it may run into trouble on specific configurationsor versions we did not encounter before. If you run into trouble using a cluster not in our list of officially supported ones, please drop us a note on the users mailing list.

A configuration file has to be valid JSON. The structure is as follows:

```
# filename: lrz.json
{
    "supermuc":
    {
        "description"                : "The SuperMUC petascale HPC cluster at LRZ.",
        "notes"                      : "Access only from registered IP addresses.",
        "schemas"                    : ["gsissh", "ssh"],
        "ssh"                        :
        {
            "job_manager_endpoint"   : "loadl+ssh://supermuc.lrz.de/",
            "filesystem_endpoint"    : "sftp://supermuc.lrz.de/"
        },
        "gsissh"                     :
        {
            "job_manager_endpoint"   : "loadl+gsissh://supermuc.lrz.de:2222/",
            "filesystem_endpoint"    : "gsisftp://supermuc.lrz.de:2222/"
        },
        "default_queue"              : "test",
        "lrms"                       : "LOADL",
        "task_launch_method"         : "SSH",
        "mpi_launch_method"          : "MPIEXEC",
        "forward_tunnel_endpoint"    : "login03",
        "global_virtenv"             : "/home/hpc/pr87be/di29sut/pilotve",
        "pre_bootstrap"              : ["source /etc/profile",
                                        "source /etc/profile.d/modules.sh",
                                        "module load python/2.7.6",
                                        "module unload mpi.ibm", "module load mpi.intel",
                                        "source /home/hpc/pr87be/di29sut/pilotve/bin/activate"
                                        ],
        "valid_roots"                : ["/home", "/gpfs/work", "/gpfs/scratch"],
        "pilot_agent"                : "radical-pilot-agent-multicore.py"
    },
    "ANOTHER_KEY_NAME":
    {
        ...
    }
}
```

The name of your file (here lrz.json) together with the name of the resource (supermuc) form the resource key which is used in the class:ComputePilotDescription resource attribute (lrz.supermuc).

All fields are mandatory, unless indicated otherwise below.

- **description**: a human readable description of the resource
- **notes**: information needed to form valid pilot descriptions, such as which parameter are required, etc.
- **schemas**: allowed values for the access_schema parameter of the pilot description. The first schema in the list is used by default. For each schema, a subsection is needed which specifies job_manager_endpoint and filesystem_endpoint.
- **job_manager_endpoint**: access url for pilot submission (interpreted by SAGA)

- **filesystem_endpoint**: access url for file staging (interpreted by SAGA)
- **default_queue**: queue to use for pilot submission (optional)
- **lrms**: type of job management system (LOADL, LSF, PBSPRO, SGE, SLURM, TORQUE, FORK)
- **task_launch_method**: type of compute node access (required for non-MPI units: SSH,'APRUN' or LOCAL)
- **mpi_launch_method**: type of MPI support (required for MPI units: MPIRUN, MPIEXEC, APRUN, IBRUN or POE)
- **python_interpreter**: path to python (optional)
- **pre_bootstrap**: list of commands to execute for initialization (optional)
- **valid_roots**: list of shared file system roots (optional). Pilot sandboxes must lie under these roots.
- **pilot_agent**: type of pilot agent to use (radical-pilot-agent-multicore.py)
- **forward_tunnel_endpoint**: name of host which can be used to create ssh tunnels from the compute nodes to the outside world (optional)

Several configuration files are part of the RADICAL-Pilot installation, and live under radical/pilot/configs/.

CHAPTER 9

List of Pre-Configured Resources

The following list of resources are supported by the underlying layers of ExTASY.

Note: To configure your applications to run on these machines, you would need to add entries to your kernel definitions and specify the environment to be loaded for execution, executable, arguments, etc.

9.1 RESOURCE_FUTUREGRID

9.1.1 BRAVO

FutureGrid Hewlett-Packard ProLiant compute cluster (https://futuregrid.github.io/manual/hardware.html).

- **Resource label** : `futuregrid.bravo`
- **Raw config** : `resource_futuregrid.json`
- **Note** : Works only up to 64 cores, beyond that Torque configuration is broken.
- **Default values** for ComputePilotDescription attributes:
- `queue : bravo`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.1.2 INDIA

The FutureGrid 'india' cluster (https://futuregrid.github.io/manual/hardware.html).

- **Resource label** : `futuregrid.india`
- **Raw config** : `resource_futuregrid.json`
- **Default values** for ComputePilotDescription attributes:
- `queue : batch`
- `sandbox : $HOME`
- `access_schema : ssh`

- **Available schemas** : `ssh`

9.1.3 ECHO

FutureGrid Supermicro ScaleMP cluster (https://futuregrid.github.io/manual/hardware.html).

- **Resource label** : `futuregrid.echo`
- **Raw config** : `resource_futuregrid.json`
- **Note** : Untested
- **Default values** for ComputePilotDescription attributes:
- `queue : echo`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.1.4 XRAY

FutureGrid Cray XT5m cluster (https://futuregrid.github.io/manual/hardware.html).

- **Resource label** : `futuregrid.xray`
- **Raw config** : `resource_futuregrid.json`
- **Note** : One needs to add 'module load torque' to ~/.profile on xray.
- **Default values** for ComputePilotDescription attributes:
- `queue : batch`
- `sandbox : /scratch/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.1.5 XRAY_CCM

FutureGrid Cray XT5m cluster in Cluster Compatibility Mode (CCM) (https://futuregrid.github.io/manual/hardware.html).

- **Resource label** : `futuregrid.xray_ccm`
- **Raw config** : `resource_futuregrid.json`
- **Note** : One needs to add 'module load torque' to ~/.profile on xray.
- **Default values** for ComputePilotDescription attributes:
- `queue : ccm_queue`
- `sandbox : /scratch/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.1.6 DELTA

FutureGrid Supermicro GPU cluster (https://futuregrid.github.io/manual/hardware.html).

- **Resource label** : `futuregrid.delta`
- **Raw config** : `resource_futuregrid.json`
- **Note** : Untested.
- **Default values** for ComputePilotDescription attributes:
- `queue : delta`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.2 RESOURCE_ORNL

9.2.1 TITAN

The Cray XK7 supercomputer located at the Oak Ridge Leadership Computing Facility (OLCF), (https://www.olcf.ornl.gov/titan/)

- **Resource label** : `ornl.titan`
- **Raw config** : `resource_ornl.json`
- **Note** : Requires the use of an RSA SecurID on every connection.
- **Default values** for ComputePilotDescription attributes:
- `queue : batch`
- `sandbox : $MEMBERWORK/`groups | cut -d' ' -f2`
- `access_schema : ssh`
- **Available schemas** : `ssh, local, go`

9.3 RESOURCE_IU

9.3.1 BIGRED2

Indiana University's Cray XE6/XK7 cluster (https://kb.iu.edu/d/bcqt).

- **Resource label** : `iu.bigred2`
- **Raw config** : `resource_iu.json`
- **Default values** for ComputePilotDescription attributes:
- `queue : None`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.3.2 BIGRED2_CCM

Indiana University's Cray XE6/XK7 cluster in Cluster Compatibility Mode (CCM) (https://kb.iu.edu/d/bcqt).

- **Resource label** : `iu.bigred2_ccm`
- **Raw config** : `resource_iu.json`
- **Default values** for ComputePilotDescription attributes:
- `queue : None`
- `sandbox : /N/dc2/scratch/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.4 RESOURCE_RADICAL

9.4.1 TUTORIAL

Our private tutorial VM on EC2

- **Resource label** : `radical.tutorial`
- **Raw config** : `resource_radical.json`
- **Default values** for ComputePilotDescription attributes:
- `queue : batch`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh, local`

9.5 RESOURCE_RICE

9.5.1 DAVINCI

The DAVinCI Linux cluster at Rice University (https://docs.rice.edu/confluence/display/ITDIY/Getting+Started+on+DAVinCI).

- **Resource label** : `rice.davinci`
- **Raw config** : `resource_rice.json`
- **Note** : DAVinCI compute nodes have 12 or 16 processor cores per node.
- **Default values** for ComputePilotDescription attributes:
- `queue : parallel`
- `sandbox : $SHARED_SCRATCH/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.5.2 BIOU

The Blue BioU Linux cluster at Rice University (https://docs.rice.edu/confluence/display/ITDIY/Getting+Started+on+Blue+BioU).

- **Resource label** : `rice.biou`
- **Raw config** : `resource_rice.json`
- **Note** : Blue BioU compute nodes have 32 processor cores per node.
- **Default values** for ComputePilotDescription attributes:
- `queue : serial`
- `sandbox : $SHARED_SCRATCH/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.6 RESOURCE_XSEDE

9.6.1 LONESTAR

The XSEDE 'Lonestar' cluster at TACC (https://www.tacc.utexas.edu/resources/hpc/lonestar).

- **Resource label** : `xsede.lonestar`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh`

9.6.2 STAMPEDE_YARN

The XSEDE 'Stampede' cluster at TACC (https://www.tacc.utexas.edu/stampede/).

- **Resource label** : `xsede.stampede_yarn`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : $WORK`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh, go`

9.6.3 STAMPEDE

The XSEDE 'Stampede' cluster at TACC (https://www.tacc.utexas.edu/stampede/).

- **Resource label** : `xsede.stampede`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : $WORK`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh, go`

9.6.4 BLACKLIGHT

The XSEDE 'Blacklight' cluster at PSC (https://www.psc.edu/index.php/computing-resources/blacklight).

- **Resource label** : `xsede.blacklight`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : batch`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh`

9.6.5 COMET

The Comet HPC resource at SDSC 'HPC for the 99%' (http://www.sdsc.edu/services/hpc/hpc_systems.html#comet).

- **Resource label** : `xsede.comet`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : compute`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh`

9.6.6 SUPERMIC

SuperMIC (pronounced 'Super Mick') is Louisiana State University's (LSU) newest supercomputer funded by the National Science Foundation's (NSF) Major Research Instrumentation (MRI) award to the Center for Computation & Technology. (https://portal.xsede.org/lsu-supermic)

- **Resource label** : `xsede.supermic`
- **Raw config** : `resource_xsede.json`
- **Note** : Partially allocated through XSEDE. Primary access through GSISSH. Allows SSH key authentication too.
- **Default values** for ComputePilotDescription attributes:
- `queue : workq`
- `sandbox : /work/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh`

9.6.7 COMET_ORTE

The Comet HPC resource at SDSC 'HPC for the 99%' (http://www.sdsc.edu/services/hpc/hpc_systems.html#comet).

- **Resource label** : `xsede.comet_orte`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : compute`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh`

9.6.8 TRESTLES

The XSEDE 'Trestles' cluster at SDSC (http://www.sdsc.edu/us/resources/trestles/).

- **Resource label** : `xsede.trestles`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh`

9.6.9 GORDON

The XSEDE 'Gordon' cluster at SDSC (http://www.sdsc.edu/us/resources/gordon/).

- **Resource label** : `xsede.gordon`
- **Raw config** : `resource_xsede.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh, gsissh`

9.7 RESOURCE_LOCAL

9.7.1 LOCALHOST_YARN

Your local machine.

- **Resource label** : `local.localhost_yarn`
- **Raw config** : `resource_local.json`
- **Note** : To use the ssh schema, make sure that ssh access to localhost is enabled.
- **Default values** for ComputePilotDescription attributes:
- `queue : None`
- `sandbox : $HOME`
- `access_schema : local`
- **Available schemas** : `local, ssh`

9.7.2 LOCALHOST_ANACONDA

Your local machine.

- **Resource label** : `local.localhost_anaconda`
- **Raw config** : `resource_local.json`
- **Note** : To use the ssh schema, make sure that ssh access to localhost is enabled.
- **Default values** for ComputePilotDescription attributes:
- `queue : None`
- `sandbox : $HOME`
- `access_schema : local`
- **Available schemas** : `local, ssh`

9.7.3 LOCALHOST

Your local machine.

- **Resource label** : `local.localhost`
- **Raw config** : `resource_local.json`
- **Note** : To use the ssh schema, make sure that ssh access to localhost is enabled.
- **Default values** for ComputePilotDescription attributes:
- `queue : None`
- `sandbox : $HOME`
- `access_schema : local`
- **Available schemas** : `local, ssh`

9.8 RESOURCE_NCAR

9.8.1 YELLOWSTONE

The Yellowstone IBM iDataPlex cluster at UCAR (https://www2.cisl.ucar.edu/resources/yellowstone).

- **Resource label** : `ncar.yellowstone`
- **Raw config** : `resource_ncar.json`
- **Note** : We only support one concurrent CU per node currently.
- **Default values** for ComputePilotDescription attributes:
- `queue : premium`
- `sandbox : $HOME`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.9 RESOURCE_STFC

9.9.1 JOULE

The STFC Joule IBM BG/Q system (http://community.hartree.stfc.ac.uk/wiki/site/admin/home.html)

- **Resource label** : `stfc.joule`
- **Raw config** : `resource_stfc.json`
- **Note** : This currently needs a centrally administered outbound ssh tunnel.
- **Default values** for ComputePilotDescription attributes:
- `queue : prod`
- `sandbox : $HOME`
- `access_schema : ssh`

- **Available schemas** : `ssh`

9.10 RESOURCE_EPSRC

9.10.1 ARCHER

The EPSRC Archer Cray XC30 system (https://www.archer.ac.uk/)

- **Resource label** : `epsrc.archer`
- **Raw config** : `resource_epsrc.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : standard`
- `sandbox : /work/`id -gn`/`id -gn`/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.10.2 ARCHER_ORTE

The EPSRC Archer Cray XC30 system (https://www.archer.ac.uk/)

- **Resource label** : `epsrc.archer_orte`
- **Raw config** : `resource_epsrc.json`
- **Note** : Always set the `project` attribute in the ComputePilotDescription or the pilot will fail.
- **Default values** for ComputePilotDescription attributes:
- `queue : standard`
- `sandbox : /work/`id -gn`/`id -gn`/$USER`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.11 RESOURCE_DAS4

9.11.1 FS2

The Distributed ASCI Supercomputer 4 (http://www.cs.vu.nl/das4/).

- **Resource label** : `das4.fs2`
- **Raw config** : `resource_das4.json`
- **Default values** for ComputePilotDescription attributes:
- `queue : all.q`
- `sandbox : $HOME`

- `access_schema : ssh`
- **Available schemas** : `ssh`

9.12 RESOURCE_NCSA

9.12.1 BW_CCM

The NCSA Blue Waters Cray XE6/XK7 system in CCM (https://bluewaters.ncsa.illinois.edu/)

- **Resource label** : `ncsa.bw_ccm`
- **Raw config** : `resource_ncsa.json`
- **Note** : Running 'touch .hushlogin' on the login node will reduce the likelihood of prompt detection issues.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : /scratch/sciteam/$USER`
- `access_schema : gsissh`
- **Available schemas** : `gsissh`

9.12.2 BW

The NCSA Blue Waters Cray XE6/XK7 system (https://bluewaters.ncsa.illinois.edu/)

- **Resource label** : `ncsa.bw`
- **Raw config** : `resource_ncsa.json`
- **Note** : Running 'touch .hushlogin' on the login node will reduce the likelihood of prompt detection issues.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : /scratch/sciteam/$USER`
- `access_schema : gsissh`
- **Available schemas** : `gsissh`

9.12.3 BW_APRUN

The NCSA Blue Waters Cray XE6/XK7 system (https://bluewaters.ncsa.illinois.edu/)

- **Resource label** : `ncsa.bw_aprun`
- **Raw config** : `resource_ncsa.json`
- **Note** : Running 'touch .hushlogin' on the login node will reduce the likelihood of prompt detection issues.
- **Default values** for ComputePilotDescription attributes:
- `queue : normal`
- `sandbox : /scratch/sciteam/$USER`

- `access_schema : gsissh`
- **Available schemas** : `gsissh`

9.13 RESOURCE_NERSC

9.13.1 EDISON_CCM

The NERSC Edison Cray XC30 in Cluster Compatibility Mode (https://www.nersc.gov/users/computational-systems/edison/)

- **Resource label** : `nersc.edison_ccm`
- **Raw config** : `resource_nersc.json`
- **Note** : For CCM you need to use special **ccm_** queues.
- **Default values** for ComputePilotDescription attributes:
- `queue : ccm_queue`
- `sandbox : $SCRATCH`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.13.2 EDISON

The NERSC Edison Cray XC30 (https://www.nersc.gov/users/computational-systems/edison/)

- **Resource label** : `nersc.edison`
- **Raw config** : `resource_nersc.json`
- **Note** :
- **Default values** for ComputePilotDescription attributes:
- `queue : regular`
- `sandbox : $SCRATCH`
- `access_schema : ssh`
- **Available schemas** : `ssh, go`

9.13.3 HOPPER

The NERSC Hopper Cray XE6 (https://www.nersc.gov/users/computational-systems/hopper/)

- **Resource label** : `nersc.hopper`
- **Raw config** : `resource_nersc.json`
- **Note** :
- **Default values** for ComputePilotDescription attributes:
- `queue : regular`
- `sandbox : $SCRATCH`

- `access_schema : ssh`
- **Available schemas** : `ssh, go`

9.13.4 HOPPER_APRUN

The NERSC Hopper Cray XE6 (https://www.nersc.gov/users/computational-systems/hopper/)

- **Resource label** : `nersc.hopper_aprun`
- **Raw config** : `resource_nersc.json`
- **Note** : Only one CU per node in APRUN mode
- **Default values** for ComputePilotDescription attributes:
- `queue : regular`
- `sandbox : $SCRATCH`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.13.5 HOPPER_CCM

The NERSC Hopper Cray XE6 in Cluster Compatibility Mode (https://www.nersc.gov/users/computational-systems/hopper/)

- **Resource label** : `nersc.hopper_ccm`
- **Raw config** : `resource_nersc.json`
- **Note** : For CCM you need to use special **ccm_** queues.
- **Default values** for ComputePilotDescription attributes:
- `queue : ccm_queue`
- `sandbox : $SCRATCH`
- `access_schema : ssh`
- **Available schemas** : `ssh`

9.13.6 EDISON_APRUN

The NERSC Edison Cray XC30 (https://www.nersc.gov/users/computational-systems/edison/)

- **Resource label** : `nersc.edison_aprun`
- **Raw config** : `resource_nersc.json`
- **Note** : Only one CU per node in APRUN mode
- **Default values** for ComputePilotDescription attributes:
- `queue : regular`
- `sandbox : $SCRATCH`
- `access_schema : ssh`
- **Available schemas** : `ssh, go`

9.14 RESOURCE_LRZ

9.14.1 SUPERMUC

The SuperMUC petascale HPC cluster at LRZ, Munich (http://www.lrz.de/services/compute/supermuc/).

- **Resource label** : `lrz.supermuc`
- **Raw config** : `resource_lrz.json`
- **Note** : Default authentication to SuperMUC uses X509 and is firewalled, make sure you can gsissh into the machine from your registered IP address. Because of outgoing traffic restrictions your MongoDB needs to run on a port in the range 20000 to 25000.
- **Default values** for ComputePilotDescription attributes:
- `queue : test`
- `sandbox : $HOME`
- `access_schema : gsissh`
- **Available schemas** : `gsissh, ssh`

CHAPTER 10

Troubleshooting

Some issues that you might face during the execution are discussed here.

10.1 Execution fails with "Couldn't read packet: Connection reset by peer"

You encounter the following error when running any of the extasy workflows:

```
...
########################
##       ERROR        ##
########################
Pilot 54808707f8cdba339a7204ce has FAILED. Can't recover.
Pilot log: [u'Pilot launching failed: Insufficient system resources: Insufficient system resources: r
...
```

TO fix this, create a file `~/.saga/cfg` in your home directory and add the following two lines:

```
[saga.utils.pty]
ssh_share_mode = no
```

This switches the SSH transfer layer into "compatibility" mode which should address the "Connection reset by peer" problem.

10.2 Configuring SSH Access

From a terminal from your local machine, setup a key pair with your email address.

```
    $ ssh-keygen -t rsa -C "name@email.com"

    Generating public/private rsa key pair.
    Enter file in which to save the key (/home/user/.ssh/id_rsa): [Enter]
    Enter passphrase (empty for no passphrase): [Passphrase]
    Enter same passphrase again: [Passphrase]
    Your identification has been saved in /home/user/.ssh/id_rsa.
    Your public key has been saved in /home/user/.ssh/id_rsa.pub.
    The key fingerprint is:
    03:d4:c4:6d:58:0a:e2:4a:f8:73:9a:e8:e3:07:16:c8 your@email.ac.uk
    The key's randomart image is:
    +--[ RSA 2048]----+
```

```
|     . ...+o++++. |
| . . . =o..       |
|+ . . .......o o  |
|oE .   .          |
|o =     .   S     |
|.    +.+     .    |
|.  oo             |
|.  .              |
| ..               |
+------------------+
```

Next you need to transfer it to the remote machine.

To transfer to Stampede,

```
$cat ~/.ssh/id_rsa.pub | ssh username@stampede.tacc.utexas.edu 'cat - >> ~/.ssh/authorized_keys'
```

To transfer to Archer,

```
cat ~/.ssh/id_rsa.pub | ssh username@login.archer.ac.uk 'cat - >> ~/.ssh/authorized_keys'
```

10.3 Error: Permission denied (publickey,keyboard-interactive) in AGENT.STDERR

The Pilot does not start running and goes to the 'Done' state directly from 'PendingActive'. Please check the AGENT.STDERR file for "Permission denied (publickey,keyboard-interactive)" .

```
Permission denied (publickey,keyboard-interactive).
kill: 19932: No such process
```

You require to setup passwordless, intra-node SSH access. Although this is default in most HPC clusters, this might not be the case always.

On the head-node, run:

```
cd ~/.ssh/
ssh-keygen -t rsa
```

Do not enter a passphrase. The result should look like this:

```
Generating public/private rsa key pair.

Enter file in which to save the key (/home/e290/e290/oweidner/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/e290/e290/oweidner/.ssh/id_rsa.
Your public key has been saved in /home/e290/e290/oweidner/.ssh/id_rsa.pub.
The key fingerprint is:
73:b9:cf:45:3d:b6:a7:22:72:90:28:0a:2f:8a:86:fd oweidner@eslogin001
The key's randomart image is:
+--[ RSA 2048]----+
|     . ...+o++++. |
| . . . =o..       |
|+ . . .......o o  |
|oE .   .          |
|o =     .   S     |
|.    +.+     .    |
```

```
    |.   oo            |
    |.   .             |
    | ..               |
    +-----------------+
```

Next, you need to add this key to the authorized_keys file.

```
    cat id_rsa.pub >> ~/.ssh/authorized_keys
```

This should be all. Next time you run radical.pilot, you shouldn't see that error message anymore.

10.4 Error: Couldn't create new session

If you get an error similar to,

```
An error occurred: Couldn't create new session (database URL 'mongodb://extasy:extasyproject@extasy-d
Exception triggered, no session created, exiting now...
```

This means no session was created, mostly due to error in the MongoDB URL that is present in the resource configuration file. Please check the URL that you have used. If the URL is correct, you should check the system on which the MongoDB is hosted.

10.5 Error: Prompted for unkown password

If you get an error similar to,

```
An error occurred: prompted for unknown password (username@stampede.tacc.utexas.edu's password: ) (/e
```

You should check the username that is present in the resource configuration file. If the username is correct, you should check if you have a passwordless login set up for the target machine. You can check this by simply attempting a login to the target machine, if this attempt requires a password, you need to set up a passwordless login to use ExTASY.

10.6 Error: Pilot has FAILED. Can't recover

If you get an error similar to,

```
ExTASY version :  0.1.3-beta-15-g9e16ce7
Session UID: 55102e9023769c19e7c8a84e
Pilot UID       : 55102e9123769c19e7c8a850
[Callback]: ComputePilot '55102e9123769c19e7c8a850' state changed to Launching.
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Loading kernel configurations from /experiments/extasy/lib/python2.7/site-packages/radical/ensemblemd
Preprocessing stage ....
[Callback]: ComputePilot '55102e9123769c19e7c8a850' state changed to Failed.
#######################
##       ERROR       ##
```

```
#######################
Pilot 55102e9123769c19e7c8a850 has FAILED. Can't recover.
Pilot log: [<radical.pilot.logentry.Logentry object at 0x7f41f8043a10>, <radical.pilot.logentry.Loger
Execution was interrupted
Closing session, exiting now ...
```

This generally means either the Allocation ID or Queue name present in the resource configuration file is incorrect. If this is not the case, please re-run the experiment with the environment variables EXTASY_DEBUG=True, SAGA_VERBOSE=DEBUG, RADICAL_PILOT_VERBOSE=DEBUG. Example,

```
EXTASY_DEBUG=True SAGA_VERBOSE=DEBUG RADICAL_PILOT_VERBOSE=DEBUG extasy --RPconfig stampede.rcfg --Ko
```

This should generate a more verbose output. You may look at this verbose output for errors or create a ticket with this log here)

10.7 Couldn't send packet: Broken pipe

If you get an error similar to,

```
2015:03:30 16:05:07 radical.pilot.MainProcess: [DEBUG   ] read : [   19] [  159] ( ls /work/e290/e290
2015:03:30 16:05:08 radical.pilot.MainProcess: [ERROR   ] Output transfer failed: read from process f
sftp>  ls /work/e290/e290/e290ib/radical.pilot.sandbox/pilot-55196431d7bf7579ecc ^H3f080/unit-551965f
Couldn't send packet: Broken pipe
```

This is mostly because of an older version of sftp/scp being used. This can be fixed by setting an environment variable `SAGA_PTY_SSH_SHAREMODE` to `no`.

```
export SAGA_PTY_SSH_SHAREMODE=no
```

10.8 Writing a Custom Resource Configuration File

If you want to use RADICAL-Pilot with a resource that is not in any of the provided configuration files, you can write your own, and drop it in $HOME/.radical/pilot/configs/<your_site>.json.

Note: Be advised that you may need system admin level knowledge for the target cluster to do so. Also, while RADICAL-Pilot can handle very different types of systems and batch system, it may run into trouble on specific configurationsor versions we did not encounter before. If you run into trouble using a cluster not in our list of officially supported ones, please drop us a note on the users mailing list.

A configuration file has to be valid JSON. The structure is as follows:

```
    # filename: lrz.json
    {
        "supermuc":
    {
        "description"                 : "The SuperMUC petascale HPC cluster at LRZ.",
            "notes"                   : "Access only from registered IP addresses.",
            "schemas"                 : ["gsissh", "ssh"],
            "ssh"                     :
            {
            "job_manager_endpoint"    : "loadl+ssh://supermuc.lrz.de/",
            "filesystem_endpoint"     : "sftp://supermuc.lrz.de/"
```

```
            },
            "gsissh"                        :
            {
                "job_manager_endpoint"     : "loadl+gsissh://supermuc.lrz.de:2222/",
            "filesystem_endpoint"      : "gsisftp://supermuc.lrz.de:2222/"
            },
            "default_queue"                 : "test",
            "lrms"                          : "LOADL",
            "task_launch_method"            : "SSH",
            "mpi_launch_method"             : "MPIEXEC",
            "forward_tunnel_endpoint"       : "login03",
            "global_virtenv"                : "/home/hpc/pr87be/di29sut/pilotve",
            "pre_bootstrap"                 : ["source /etc/profile",
                                               "source /etc/profile.d/modules.sh",
                                               "module load python/2.7.6",
                                               "module unload mpi.ibm", "module load mpi.intel",
                                               "source /home/hpc/pr87be/di29sut/pilotve/bin/activate"
                                               ],
            "valid_roots"                   : ["/home", "/gpfs/work", "/gpfs/scratch"],
            "pilot_agent"                   : "radical-pilot-agent-multicore.py"
    },
    "ANOTHER_KEY_NAME":
    {
            ...
    }
    }
```

The name of your file (here lrz.json) together with the name of the resource (supermuc) form the resource key which is used in the class:ComputePilotDescription resource attribute (lrz.supermuc).

All fields are mandatory, unless indicated otherwise below.

- **description**: a human readable description of the resource
- **notes**: information needed to form valid pilot descriptions, such as which parameter are required, etc.
- **schemas**: allowed values for the access_schema parameter of the pilot description. The first schema in the list is used by default. For each schema, a subsection is needed which specifies job_manager_endpoint and filesystem_endpoint.
- **job_manager_endpoint**: access url for pilot submission (interpreted by SAGA)
- **filesystem_endpoint**: access url for file staging (interpreted by SAGA)
- **default_queue**: queue to use for pilot submission (optional)
- **lrms**: type of job management system (LOADL, LSF, PBSPRO, SGE, SLURM, TORQUE, FORK)
- **task_launch_method**: type of compute node access (required for non-MPI units: SSH,'APRUN' or LOCAL)
- **mpi_launch_method**: type of MPI support (required for MPI units: MPIRUN, MPIEXEC, APRUN, IBRUN or POE)
- **python_interpreter**: path to python (optional)
- **pre_bootstrap**: list of commands to execute for initialization (optional)
- **valid_roots**: list of shared file system roots (optional). Pilot sandboxes must lie under these roots.
- **pilot_agent**: type of pilot agent to use (radical-pilot-agent-multicore.py)
- **forward_tunnel_endpoint**: name of host which can be used to create ssh tunnels from the compute nodes to the outside world (optional)

Several configuration files are part of the RADICAL-Pilot installation, and live under radical/pilot/configs/.

CHAPTER 11

Miscellaneous

11.1 List of Supported Kernels

- md.amber
- md.coco
- md.gromacs
- md.lsdmap
- md.mmpbsa
- md.namd
- md.post_lsdmap
- md.pre_coam_loop
- md.pre_grlsd_loop
- md.pre_lsdmap
- md.re_exchange
- md.tleap
- misc.ccount
- misc.chksum
- misc.diff
- misc.idle
- misc.levenshtein
- misc.mkfile
- misc.nop
- misc.randval

CHAPTER 12

Indices and tables

- genindex
- search