ExTASY Documentation

Release 0.1-beta

RADICAL Group at Rutgers

October 01, 2015

Contents

Github Page

https://github.com/radical-cybertools/ExTASY/tree/extasy_0.1

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

This page 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
```

Step 2 : Install ExTASY,

```
pip install --upgrade git+https://github.com/radical-cybertools/ExTASY.git@extasy_0.1#egg=radical.ens
```

To install the development version (unstable),

```
pip install --upgrade git+https://github.com/radical-cybertools/ExTASY.git@extasy_0.1#egg=radical.ens
```

```
pip install --upgrade git+https://github.com/radical-cybertools/radical.ensemblemd.mdkernels.git@mast
```

Now you should be able to print the installed version of the ExTASY module using,

```
python -c 'import radical.ensemblemd.extasy as extasy; print extasy.version'
```

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 !

CHAPTER 3

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.

3.1 Running on Stampede

3.1.1 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_0.1/tarballs/coam
tar xvfz coam-on-stampede.tar.gz
```

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.

```
REMOTE_HOST = 'xsede.stampede'  # Label/Name of the Remote Machine
UNAME       = 'username'                # Username on the Remote Machine
ALLOCATION  = 'TG-MCB090174'            # Allocation to be charged
WALLTIME    = 60                        # Walltime to be requested for the pilot
PILOTSIZE   = 16                        # Number of cores to be reserved
WORKDIR     = None                      # Working directory on the remote machine
QUEUE       = 'normal'                  # Name of the queue in the remote machine

DBURL       = 'mongodb://extasy:extasyproject@extasy-db.epcc.ed.ac.uk/radicalpilot'
```

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

```
#-------------------------Applications----------------------
simulator                 = 'Amber'          # Simulator to be loaded
analyzer                  = 'CoCo'           # Analyzer to be loaded

#-------------------------General---------------------------
num_iterations          = 4                # Number of iterations of Simulation-Analysis
start_iter              = 0                # Iteration number with which to start
num_CUs                 = 16               # Number of tasks or Compute Units
nsave                   = 2                # Iterations after which output is transfered to loc
checkfiles              = 4                # Iterations after which to test if the expected fil

#-------------------------Simulation------------------------
num_cores_per_sim_cu    = 2                # Number of cores per Simulation Compute Units
md_input_file           = './mdshort.in'   # Entire path to MD Input file - Do not use $HOME or
minimization_input_file = './min.in'       # Entire path to Minimization file - Do not use $HOM
initial_crd_file        = './penta.crd'    # Entire path to Coordinates file - Do not use $HOME
top_file                = './penta.top'    # Entire path to Topology file - Do not use $HOME or
logfile                 = 'coco.log'       # Name of the log file created by pyCoCo
atom_selection          = None             # optional atom selection string that enables pyCoCo

#-------------------------Analysis--------------------------
grid                    = '5'              # Number of points along each dimension of the CoCo
dims                    = '3'              # The number of projections to consider from the inp
```

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

Now you are can run the workload using :

If your shell is BASH,

```
EXTASY_DEBUG=True RADICAL_PILOT_VERBOSE='debug' SAGA_VERBOSE='debug' extasy --RPconfig stampede.
```

If your shell is CSH,

```
setenv EXTASY_DEBUG True
setenv RADICAL_PILOT_VERBOSE 'debug'
setenv SAGA_VERBOSE 'debug'
extasy --RPconfig stampede.rcfg --Kconfig cocoamber.wcfg |& tee extasy.log
```

A **sample output** with expected callbacks and simulation/analysis can be found at here.

Stage	Simulation	Analysis
Expected TTC/iteration	30-35 s	25-30 s

There are two stages in the execution phase - Simulation and Analysis. Execution starts with any Preprocessing that might be required on the input data and then moves to Simulation stage. In the Simulation stage, a number of tasks (num_CUs) are launched to execute on the target machine. The number of tasks set to execute depends on the PILOTSIZE, num_CUs, num_cores_per_sim_cu, the number of tasks in execution state simultaneously would be PILOTSIZE/num_cores_per_sim_cu. As each task attains 'Done' (completed) state, the remain tasks are scheduled till all the num_CUs tasks are completed.

This is followed by the Analysis stage, one task is scheduled on the target machine which takes all the cores as the PILOTSIZE to perform the analysis and returns the data required for the next iteration of the Simulation stage. As can be seen, per iteration, there are (num_CUs+1) tasks executed.

3.2 Running on Archer

3.2.1 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_0.1/tarballs/coam
tar xvfz coam-on-archer.tar.gz
```

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.

```
REMOTE_HOST = 'epsrc.archer'              # Label/Name of the Remote Machine
UNAME       = 'username'                  # Username on the Remote Machine
ALLOCATION  = 'e290'                      # Allocation to be charged
WALLTIME    = 60                          # Walltime to be requested for the pilot
PILOTSIZE   = 24                          # Number of cores to be reserved
WORKDIR     = None                        # Working directory on the remote machine
QUEUE       = 'standard'                  # Name of the queue in the remote machine

DBURL       = 'mongodb://extasy:extasyproject@extasy-db.epcc.ed.ac.uk/radicalpilot'
```

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

```
    #------------------------Applications----------------------
    simulator                 = 'Amber'          # Simulator to be loaded
    analyzer                  = 'CoCo'           # Analyzer to be loaded

    #------------------------General---------------------------
    num_iterations          = 2                  # Number of iterations of Simulation-Analysis
    start_iter              = 0                  # Iteration number with which to start
    num_CUs                 = 8                  # Number of tasks or Compute Units
    nsave                   = 1                  # Iterations after which output is transfered to loc
    checkfiles              = 4                  # Iterations after which to test if the expected fil

    #------------------------Simulation------------------------
    num_cores_per_sim_cu    = 2                  # Number of cores per Simulation Compute Units
    md_input_file           = './mdshort.in'     # Entire path to MD Input file - Do not use $HOME or
    minimization_input_file = './min.in'         # Entire path to Minimization file - Do not use $HOM
    initial_crd_file        = './penta.crd'      # Entire path to Coordinates file - Do not use $HOME
    top_file                = './penta.top'      # Entire path to Topology file - Do not use $HOME or
    logfile                 = 'coco.log'         # Name of the log file created by pyCoCo
    atom_selection          = None               # optional atom selection string that enables pyCoCo

    #------------------------Analysis--------------------------
    grid                    = '5'                # Number of points along each dimension of the CoCo
    dims                    = '3'                # The number of projections to consider from the inp
```

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

Now you are can run the workload using :

If your shell is BASH,

```
    EXTASY_DEBUG=True RADICAL_PILOT_VERBOSE='debug' SAGA_VERBOSE='debug' extasy --RPconfig archer.rc
```

If your shell is CSH,

```
    setenv EXTASY_DEBUG True
    setenv RADICAL_PILOT_VERBOSE 'debug'
    setenv SAGA_VERBOSE 'debug'
    extasy --RPconfig archer.rcfg --Kconfig cocoamber.wcfg |& tee extasy.log
```

A **sample output** with expected callbacks and simulation/analysis can be found at here.

Stage	Simulation	Analysis
Expected TTC/iteration	60-100 s	150-200 s

There are two stages in the execution phase - Simulation and Analysis. Execution starts with any Preprocessing that might be required on the input data and then moves to Simulation stage. In the Simulation stage, a number of tasks (num_CUs) are launched to execute on the target machine. The number of tasks set to execute depends on the PILOTSIZE, num_CUs, num_cores_per_sim_cu, the number of tasks in execution state simultaneously would be PILOTSIZE/num_cores_per_sim_cu. As each task attains 'Done' (completed) state, the remain tasks are scheduled till all the num_CUs tasks are completed.

This is followed by the Analysis stage, one task is scheduled on the target machine which takes all the cores as the PILOTSIZE to perform the analysis and returns the data required for the next iteration of the Simulation stage. As can be seen, per iteration, there are (num_CUs+1) tasks executed.

3.3 Understanding the Output

In the local machine, a "backup" 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
backup/  cocoamber.wcfg  mdshort.in  min.in  penta.crd  penta.top  stampede.rcfg

coam-on-stampede/backup$ 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/backup/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 "staging_area". This location is used to exchange/link/move intermediate data. The shared data is kept in "staging_area/" and the iteration specific inputs/outputs can be found in their specific folders (="staging_area/iter*").

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

3.4 CoCo/Amber Restart Mechanism

If the above examples were successful, you can go ahead try and the restart mechanism. The restart mechanism is designed to resume the experiment from one of the checkpoints that you might have made in the previous experiments.

Therefor, for a valid/successful restart scenario, data from a previous experiment needs to exist in the backup/ folder on the local machine. Restart can only be done from a checkpoint (defined by nsave in the kernel config file) made in the previous experiment.

Example,

> **Experiment 1** : num_iterations = 4, start_iter = 0, nsave = 2
>
> **Backups created** : iter1/ (after 2 iterations) , iter3/ (after 4 iterations)
>
> **Experiment 2 (restart)** : num_iterations = 2, start_iter = 4 (=start from 5th iter), nsave = 2
>
> **Note** : start_iter should match one of the previous checkpoints and start_iter should be a multiple of nsave.

If, in the first experiment, you ran 4 iterations with nsave set to 2, you will have backups created after the 2nd and 4th iteration. Once this is successful, in the second experiment, you can resume from either of the backups/checkpoints. In the above example, the experiment is resumed from the 4th iteration.

In CoCo/Amber, at every checkpoint the ncdf files from all the iterations are transferred to the local machine in order to be able to restart. You could set nsave = num_iterations to make a one time transfer after all the iterations.

Having a small checkpoint interval increases redundant data. Example,

Experiment 1 : num_iterations = 8, start_iter = 0, nsave = 2

Backups created :-

iter1/ (contains ncdf files for first 2 iters)

iter3/ (contains ncdf files for first 4 iters)

iter5/ (contains ncdf files for first 6 iters)

iter7/ (contains ncdf files for first 8 iters)

CHAPTER 4

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.

4.1 Running on Stampede

4.1.1 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_0.1/tarballs/grls
tar xvfz grlsd-on-stampede.tar.gz
```

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.

```
    REMOTE_HOST = 'xsede.stampede'  # Label/Name of the Remote Machine
    UNAME       = 'username'                  # Username on the Remote Machine
    ALLOCATION  = 'TG-MCB090174'              # Allocation to be charged
    WALLTIME    = 60                          # Walltime to be requested for the pilot
    PILOTSIZE   = 16                          # Number of cores to be reserved
    WORKDIR     = None                        # Working directory on the remote machine
    QUEUE       = 'normal'                    # Name of the queue in the remote machine

    DBURL       = 'mongodb://extasy:extasyproject@extasy-db.epcc.ed.ac.uk/radicalpilot'
```

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

```
    ##------------------------Applications----------------------
    simulator             = 'Gromacs'           # Simulator to be loaded
    analyzer              = 'LSDMap'            # Analyzer to be loaded

    #-------------------------General-------------------------------
    num_CUs               = 16                  # Number of tasks or Compute Units
    num_iterations        = 3                   # Number of iterations of Simulation-Analysis
    start_iter            = 0                   # Iteration number with which to start
    nsave                 = 2                   # # Iterations after which output is transfered to l
    checkfiles            = 4                   # Iterations after which to test if the expected fil

    #-------------------------Simulation----------------------------
    num_cores_per_sim_cu  = 1                   # Number of cores per Simulation Compute Units
    md_input_file         = './input.gro'       # Entire path to the MD Input file - Do not use $HOM
    mdp_file              = './grompp.mdp'      # Entire path to the MD Parameters file - Do not use
    top_file              = './topol.top'       # Entire path to the Topology file - Do not use $HOM
    ndx_file              = None                  # Entire path to the Index file - Do not use $HOME
    grompp_options        = None                  # Command line options for when grompp is used
    mdrun_options         = None                  # Command line options for when mdrun is used
    itp_file_loc          = None                  # Entire path to the location of .itp files - Do n
    md_output_file        = 'tmp.gro'           # Filename to be used for the simulation output

    #-------------------------Analysis------------------------------
    lsdm_config_file      = './config.ini'      # Entire path to the LSDMap configuration file - Do
    num_runs              = 1000                # Number of runs to be performed in the Selection ste
    w_file                = 'weight.w'          # Filename to be used for the weight file
    max_alive_neighbors   = '10'                # Maximum alive neighbors to be considered while rew
    max_dead_neighbors    = '1'                 # Maximum dead neighbors to be considered while rewe
```

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.

Now you are can run the workload using :

If your shell is BASH,

```
    EXTASY_DEBUG=True RADICAL_PILOT_VERBOSE='debug' SAGA_VERBOSE='debug' extasy --RPconfig stampede.
```

If your shell is CSH,

```
    setenv EXTASY_DEBUG True
    setenv RADICAL_PILOT_VERBOSE 'debug'
    setenv SAGA_VERBOSE 'debug'
    extasy --RPconfig stampede.rcfg --Kconfig gromacslsdmap.wcfg |& tee extasy.log
```

A **sample output** with expected callbacks and simulation/analysis can be found at here.

Stage	Simulation	Analysis
Expected TTC/iteration	50-100 s	~30 s

There are two stages in the execution phase - Simulation and Analysis. Execution starts with any Preprocessing that might be required on the input data and then moves to Simulation stage. In the Simulation stage, a number of tasks (num_CUs) are launched to execute on the target machine. The number of tasks set to execute depends on the PILOTSIZE, num_CUs, num_cores_per_sim_cu, the number of tasks in execution state simultaneously would be PILOTSIZE/num_cores_per_sim_cu. As each task attains 'Done' (completed) state, the remain tasks are scheduled till all the num_CUs tasks are completed.

This is followed by the Analysis stage, one task is scheduled on the target machine which takes all the cores as the PILOTSIZE to perform the analysis and returns the data required for the next iteration of the Simulation stage. As can be seen, per iteration, there are (num_CUs+1) tasks executed.

4.2 Running on Archer

4.2.1 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_0.1/tarballs/grls
tar xvfz grlsd-on-archer.tar.gz
```

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.

```
REMOTE_HOST = 'epsrc.archer'              # Label/Name of the Remote Machine
UNAME       = 'username'                  # Username on the Remote Machine
ALLOCATION  = 'e290'                      # Allocation to be charged
WALLTIME    = 60                          # Walltime to be requested for the pilot
PILOTSIZE   = 24                          # Number of cores to be reserved
WORKDIR     = None                        # Working directory on the remote machine
QUEUE       = 'standard'                  # Name of the queue in the remote machine
```

```
    DBURL       = 'mongodb://extasy:extasyproject@extasy-db.epcc.ed.ac.uk/radicalpilot'
```

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

```
    #------------------------Applications----------------------
    simulator             = 'Gromacs'           # Simulator to be loaded
    analyzer              = 'LSDMap'            # Analyzer to be loaded

    #-------------------------General-------------------------------
    num_CUs               = 24                  # Number of tasks or Compute Units
    num_iterations        = 2                   # Number of iterations of Simulation-Analysis
    start_iter            = 0                   # Iteration number with which to start
    nsave                 = 1                   # # Iterations after which output is transfered to l
    checkfiles            = 4                   # Iterations after which to test if the expected fil

    #-------------------------Simulation----------------------------
    num_cores_per_sim_cu = 1                    # Number of cores per Simulation Compute Units
    md_input_file         = './input.gro'       # Entire path to the MD Input file - Do not use $HOM
    mdp_file              = './grompp.mdp'      # Entire path to the MD Parameters file - Do not use
    top_file              = './topol.top'       # Entire path to the Topology file - Do not use $HOM
    ndx_file              = None                  # Entire path to the Index file - Do not use $HOME
    grompp_options        = None                  # Command line options for when grompp is used
    mdrun_options         = None                  # Command line options for when mdrun is used
    itp_file_loc          = None                  # Entire path to the location of .itp files - Do n
    md_output_file        = 'tmp.gro'           # Filename to be used for the simulation output

    #-------------------------Analysis------------------------------
    lsdm_config_file      = './config.ini'      # Entire path to the LSDMap configuration file - Do
    num_runs              = 100               # Number of runs to be performed in the Selection step
    w_file                = 'weight.w'            # Filename to be used for the weight file
    max_alive_neighbors   = '10'                  # Maximum alive neighbors to be considered while rew
    max_dead_neighbors    = '1'                   # Maximum dead neighbors to be considered while rewe
```

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.

Now you are can run the workload using :

If your shell is BASH,

```
    EXTASY_DEBUG=True RADICAL_PILOT_VERBOSE='debug' SAGA_VERBOSE='debug' extasy --RPconfig archer.rc
```

If your shell is CSH,

```
    setenv EXTASY_DEBUG True
    setenv RADICAL_PILOT_VERBOSE 'debug'
    setenv SAGA_VERBOSE 'debug'
    extasy --RPconfig archer.rcfg --Kconfig gromacslsdmap.wcfg |& tee extasy.log
```

A **sample output** with expected callbacks and simulation/analysis can be found at here.

Stage	Simulation	Analysis
Expected TTC/iteration	200-350 s	~30 s

There are two stages in the execution phase - Simulation and Analysis. Execution starts with any Preprocessing that might be required on the input data and then moves to Simulation stage. In the Simulation stage, a number of tasks (num_CUs) are launched to execute on the target machine. The number of tasks set to execute depends on

the PILOTSIZE, num_CUs, num_cores_per_sim_cu, the number of tasks in execution state simultaneously would be PILOTSIZE/num_cores_per_sim_cu. As each task attains 'Done' (completed) state, the remain tasks are scheduled till all the num_CUs tasks are completed.

This is followed by the Analysis stage, one task is scheduled on the target machine which takes all the cores as the PILOTSIZE to perform the analysis and returns the data required for the next iteration of the Simulation stage. As can be seen, per iteration, there are (num_CUs+1) tasks executed.

4.3 Understanding the Output

In the local machine, a "backup" 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
backup/  config.ini  gromacslsdmap.wcfg  grompp.mdp  input.gro  stampede.rcfg  topol.top

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

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

```
$ cd staging_area/
$ 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
```

4.4 Gromacs/LSDMap Restart Mechanism

If the above examples were successful, you can go ahead try and the restart mechanism. The restart mechanism is designed to resume the experiment from one of the checkpoints that you might have made in the previous experiments.

Therefor, for a valid/successful restart scenario, data from a previous experiment needs to exist in the backup/ folder on the local machine. Restart can only be done from a checkpoint (defined by nsave in the kernel config file) made in the previous experiment.

Example,

Experiment 1 : num_iterations = 4, start_iter = 0, nsave = 2

Backups created : iter1/ (after 2 iterations) , iter3/ (after 4 iterations)

Experiment 2 (restart) : num_iterations = 2, start_iter = 4 (=start from 5th iter), nsave = 2

Note : start_iter should match one of the previous checkpoints and start_iter should be a multiple of nsave.

If, in the first experiment, you ran 4 iterations with nsave set to 2, you will have backups created after the 2nd and 4th iteration. Once this is successful, in the second experiment, you can resume from either of the backups/checkpoints. In the above example, the experiment is resumed from the 4th iteration.

CHAPTER 5

Troubleshooting

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

5.1 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.

5.2 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'
```

5.3 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:

```
enerating 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
```

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.

5.4 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-
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.

5.5 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.

5.6 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.Logen
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 --Ke
```

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

5.7 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-5519652
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
```

5.8 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 6

Indices and tables

- genindex
- search