Integration Framework

Turbine Client PSUADE Manual

Version 2013.10.2
February 12, 2014

This material was produced under the DOE Carbon Capture Simulation Initiative (CCSI), and copyright is held by the software owners: ORISE, LANS, LLNS, LBL, PNNL, CMU, WVU, et al. The software owners and/or the U.S. Government retain ownership of all rights in the CCSI software and the copyright and patents subsisting therein. Any distribution or dissemination is governed under the terms and conditions of the CCSI Test and Evaluation License, CCSI Master Non-Disclosure Agreement, and the CCSI Intellectual Property Management Plan. No rights are granted except as expressly recited in one of the aforementioned agreements.

Table of Contents

Introduction

The Turbine Science Gateway is a web application and execution environment for running and managing scientific applications and storing and archiving results. Clients interact with the Turbine resource oriented architecture through a RESTful web interface, either directly over HTTP or using the higher-level Python client API. The Python API is designed to be easily scriptable, returning structured JSON output that can be easily consumed by other tools. Turbine is a generic solution that can be extended to process modeling and simulation applications. Currently AspenTech's AspenPlus, Aspen Custom Modeler, and Microsoft Excel applications are supported.

Turbine can be deployed on a single workstation, a cluster, and the Amazon Web Services EC2 cloud. The server-side software is Windows-based. The cluster and EC2 deployments allow for parallel application executions, which has been shown to scale up to deployments of 400 concurrent instances. This can dramatically increase application throughput and thus decrease the time to solution.

Turbine can be deployed on a single workstation, a cluster, and the Amazon Web Services EC2 cloud. The server-side software is Windows-based. The cluster and EC2 deployments allow for parallel application executions, which has been shown to scale up to deployments of 400 concurrent instances. This can dramatically increase application throughput and thus decrease the time to solution.

This guide provides detailed usage instructions for using the TurbineClient PSUADE and CSV scripts.

PSUADE Features of Turbine Client

Add PSUADE Features to a TurbineClient Config File

If you use the UQ GUI, hopefully you will never have to read this section, as the UQ GUI handles writing of the config file itself. However, for longer runs and special cases, you may need to know about this.
PSUADE and TurbineClient have a basic feature mis-match. PSUADE identifies its variable by order, while TurbineClient identifies variables by name. Furthermore, the allowable characters are not all the same. Therefore, the configuration file has three sections to tell TurbineClient the name and ordering of the PSUADE variables.
There is an additional feature of TurbineClient that causes this output section to be much more flexible, but also more complex. As part of the conversion process, you can have TurbineClient perform calculations on the output variables to produce new output variables. This will be

covered in the next tutorial.
Using PSUADE adds 3 additional sections to the TurbineClient config file. They are [Inputs], [Outputs], and [OutputsOrder]. You can see an example of the full file in python/test/psuade_test.

The [Inputs] section simply gives the <Sinter variable name>=<PSUADE order number>. So, a PSUADE input section like this:

```
INPUT
   dimension = 2
   variable 1 flash.T  =  120 180
   variable 2 flash.P  =  16  24
END
```

Should get a TurbineClient [Inputs] section like this:

```
[Inputs]
flash.T=0
flash.P=1
```

The [Outputs] and [OutputsOrder] sections are more complex. First, the PSUADE OUTPUT for reference:

```
OUTPUT
   dimension = 3
   variable 1 status
   variable 2 vapor.etOH.molefrac
   variable 3 vapor.H2O.molefrac
END
```

In the TurbineClient config file, the [Outputs] section gives each output from the simulation a python local name. This python name can be used for calculations, as described in the next section. The python names may be the same as the sinter names, but they do not have to be. Here they have different names, just “var1”, and “var2”.
Also, you should always have a “status” variable. This is the variable that tells you if the run was successful or not. ‘0’ is success, any other value is a failure. Usually ‘1’ is a simulation convergence failure and ‘-1’ is a gateway failure (the simulation never actually ran.)

```
[Outputs]
var1=vapor.etOH.molefrac
var2=vapor.H2O.molefrac
status="status"
```

The [OutputsOrder] section converts the python variables to PSUADE order variables. In this case the values are <PSUADE order> = <Python local varaible name>.

```
[OutputsOrder]
0=status
1=var1
2=var2
```

With these three section you now have a complete TurbineClient configuration to use with PSUADE.

Performing Automatic Post-Processing with PSUADE TurbineClient

This is an advanced section decribing how to have TurbineClient do simple post-processing on simulation outputs before writing them to the PSUADE file. This is useful when a simulation doesn't quite produce the output you are interested in for Uncertainty Quantification. Only the PSUADE scripts support this feature.
So, say we have a simulation we want the fraction of carbon captured from, but it does not have that value. It does have the mass of the incoming flue stream and the outgoing stream, as well as the fraction of CO2 in those streams. We can then have TurbineClient calculate the capture fraction.

We take this as out PSUADE OUTPUT section.

```
OUTPUT
   dimension = 1
   variable 1 status
   variable 2 captureFraction
END
```

TurbineClient takes the calculation as a string and runs python exec on it. The calculation must fit on a single line of Python. The calculation is defined in the [Outputs] section and the result is assigned to a python name, which can then be output as part of [OutputsOrder].

```
[Outputs]
status="status"
richmass=fluegas.mass
richco2=flugas.co2.fraction
leanmass=outgas.mass
leanco2=outgas.co2.fraction
capFrac="1-((richmass*richco2)/(leanmass*leanco2))"
```

```
[OutputsOrder]
0=status
1=capFrac
```

Performing PSUADE runs

Once you have a configuration file that supports PSUADE, actually doing a PSUADE run is straight forward. It is very similar to doing a JSON run, but somewhat safer and more streamlined.

When launching a PSUADE session, you do not need to create the session before adding jobs to it. The turbine_psuade_launch script does it all for you. The turbine_psuade_launch script also breaks the job group up into groups of 500 so you can see the launch progressing.
Here is the launch of a 1000 job run. Note the session GUID is availible both before and after the batch printouts.

The launch script takes 2 arguments:

1. A generated, but not run, PSUADE file (psuade.dat) see the PSUADE documentation for more info.
2. The TurbineClient config file, with PSUADE sections.

```
% scripts/turbine_psuade_launch psuade.dat config.txt c40a7a7c-
d904-40c0-a82b-154641c408f2
Num batches:  60
batch: 0  start: 0  end: 500  size:  500
batch: 1  start: 500  end: 1000  size:  500
c40a7a7c-d904-40c0-a82b-154641c408f2
```

After a while you may want to check how your jobs are doing. This is the same as is seen earlier in the JSON run tutorial.

```
% turbine_session_status "c40a7a7c-d904-40c0-a82b-154641c408f2"
config.txt
{"create":0,"running":100,"setup":10,"submit":800,"pause":0,"fin
ished":0,"error":10,"cancel":0,"success":80,"terminate":0}
```

From this we can see that:
800 Runs are waiting on the submit queue
80 Runs have completed successfully
10 Runs failed with an error
100 Runs are currently running
10 Runs are in setup. (It is being loaded, or the simulator is starting up. The Simulation has not actually started yet.)

After a while longer, we find that the jobs have completed:

```
% turbine_session_status "c40a7a7c-d904-40c0-a82b-154641c408f2"
config.txt
{"create":0,"running":0,"setup":0,"submit":0,"pause":0,"finished
":0,"error":100,"cancel":0,"success":900,"terminate":0}
```

Next we should get the results down from the gateway. turbine_psuade_get_results take four arguments.

1. The session GUID
2. The PSUADE input file the job was launched from. This is required for TurbineClient to correctly create the PSUADE output file. TurbineClient will also use this to check that the psuade inputs match the inputs returned from the server.
3. The filename to write the PSUADE outputs to. This file is the same as the psuadeinput file with the outputs filled in.
4. The TurbineClient configuration file.

```
% turbine_psuade_get_results "c40a7a7c-d904-40c0-a82b-
154641c408f2" psuade.dat psuade.out config.txt
```

So, the results of our run are now in the psuade.out file. You can use psuade to filter out the failed runs, or invalidate the runs that failed due to gateway issues. (That way you can rerun those failures by running turbine_psuade_launch with the psuade.out file as the input file. It will only launch the invalidated runs. See the PSUADE manual for further info, or email Jim Leek at leek2@llnl.gov.)

CSV Features of Turbine Client

Performing .csv runs

CSV support was added to allow the ALAMO optimization framework to use the gateway, but .csv is a common format, and is useful for other codes. The CSV support does not require any additional configuration beyond the basic TurbineClient configuration file, but also does not support any automated post-processing of the sort PSUADE TurbineClient does.
The CSV support also has one other major gotcha. The outputs are written as a modification to the file that give the output variable names. This means you should probably save a copy of this file, just in case.
The files for this tutorial may be found in the python/test/csv_test directory.
This tutorial assumes you have already modified the config.txt file to point to your gateway, and assumes your gateway already has the Flash_ACM simulation installed.

Here are the contents of the inputs.csv file:

```
flash.T,flash.P
142,17.1
140,18.1
```

Each column is an input variable. The first row gives the name of the input variable, each subsequent row is a run to do in the session.
Here are the contents of the outputs.csv file BEFORE running.

```
status,vapor.etOH.molefrac,vapor.H2O.molefrac
```

It is a single line giving the names of the output variables we expect to get from the simulation. Notice the first one is a status variable. The returns an integer indicating if the job passed or failed. 0 means the run passed, and other value is a failure.

First, make a copy of the outputs.csv file to actually run with. Then launch the run.

```
% cp outputs.csv outputs1.csv
% turbine_csv_launch inputs.csv config.txt
0c8d41ac-8b57-4f85-9e97-dc21dad51c8d
Num batches:  1
batch: 0  start: 0  end: 500  size:  2
0c8d41ac-8b57-4f85-9e97-dc21dad51c8d
```

Next, we check the session. Oh, it's already complete.

```
% turbine_session_status
0c8d41ac-8b57-4f85-9e97-dc21dad51c8d config.txt
{"pause": 0, "success": 2, "setup": 0, "terminate":
0, "running": 0, "submit": 0, "finished": 0, "error":
0, "cancel": 0, "create": 0}
```

Pull down the results into the output1.csv file we made earlier:

```
% turbine_csv_get_results
0c8d41ac-8b57-4f85-9e97-dc21dad51c8d inputs.csv outputs1.csv
config.txt
Jobs written to outputs.csv

% cat outputs1.csv
status,vapor.etOH.molefrac,vapor.H2O.molefrac
0,0.84003516847,0.15996483153
0,0.857740075078,0.142259924922
```

Support

ccsi-support@acceleratecarboncapture.org

Appendix A: PSUADE Script Reference

A.1 Psuade

A.1.1 turbine_psuade_doall

```
Usage: turbine_psuade_doall PSUDADE_IN PSUADE_OUT CONFIG_FILE

Takes a PSUADE file and submits all the jobs contained therein
as a new
session. Waits for it to return. Writes results to a new PSUADE
file.  (To
avoid overwriting the original) If a sessionId is supplied with
-s, it is
assumed the create and submit process has already happened, so
it simply
blocks waiting for results.

Options:
  -h, --help            show this help message and exit
  -p PAGE, --page=PAGE  page number
  -r RPP, --rpp=RPP     results per page
  -v, --verbose         verbose output
```

A.1.2 turbine_psuade_get_results

```
Usage: turbine_psuade_get_results SESSIONID PSUADE_IN PSUADE_OUT
CONFIG_FILE

 Gets all jobs from the session and writes them to a PSUADE file

Options:
  -h, --help            show this help message and exit
```

```
  -p PAGE, --page=PAGE  page number
  -r RPP, --rpp=RPP     results per page
  -v, --verbose         verbose output
```

D.1.3 turbine_psuade_launch

```
Usage: turbine_psuade_launch PSUADE_IN CONFIG_FILE

Takes a PSUADE file and submits all the jobs contained therein
as a new
session. Returns immediatly.

Options:
  -h, --help  show this help message and exit
```

D.1.4 turbine_psuade_stop

```
Usage: turbine_psuade_stop SESSIONID  CONFIG_FILE

session resource utility, stops jobs in session in state submit.
Works for
both local and gateway runs, so it's useful for the GUI.

Options:
  -h, --help  show this help message and exit
```

D.1.5 turbine_psuade_unfinished

```
Usage: turbine_psuade_unfinished [options] SESSIONID CONFIG_FILE

Reports how many jobs remain unfinished in the session (Useful
for UQ GUI)

Options:
  -h, --help  show this help message and exit
```

D.2 CSV

D.2.1 turbine_csv_get_results

```
Usage: turbine_csv_get_results SESSIONID CSV_IN CSV_OUT
CONFIG_FILE

 Gets all jobs from the session and writed them to a CSV file

Options:
  -h, --help  show this help message and exit
```

D.2.2 turbine_csv_launch

```
Usage: turbine_csv_launch CSV_IN CONFIG_FILE

Takes a CSV file and submits all the jobs contained therein as a
new session.
Returns immediatly.

Options:
  -h, --help  show this help message and exit
```