Table of Contents

1 Introduction 2

2 Installation 3

3 Using MaDaTS 4

3.1 Defining the Storage Hierarchy 4

3.2 Describing a Workflow for MaDaTS 4

3.3 Python Program using MaDaTS 6

3.3.1 Example 7

4 Execution 9

4.1 Using the Command-line 9

4.2 Using the API 9

5 MaDaTS API 10

5.1 Task 10

5.1.1 Task() 10

5.1.2 Attributes 10

5.2 VirtualDataObject 11

5.2.1 VirtualDataObject() 11

5.2.2 add_producer() 11

5.2.3 add_consumer() 11

5.2.4 Attributes 12

5.3 VirtualDataSpace 12

5.3.1 VirtualDataSpace() 12

5.3.2 map() 12

5.3.3 add() 13

5.3.4 copy() 13

5.3.5 replace() 13

5.3.6 delete() 14

5.3.7 Attributes 14

5.4 Data and Workflow Management 14

5.4.1 map() 14

5.4.2 manage() 15

5.4.3 query() 15

5.4.4 validate() 16

1 Introduction

The storage hierarchy on High Performance Computing (HPC) systems is getting deeper, driven by new technologies (NVRAMs, SSDs etc.) and the need to minimize I/O costs. The additional storage tiers introduce more complexities in workflow and data management that are often handled separately by the users of an HPC system.

Workflow tools have limited support for managing the input, output and intermediate data. The data elements of a workflow are often managed by the user through scripts or other ad-hoc mechanisms. MaDaTS (Managing Data on Tiered Storage for Scientific Workflows) is a Python library and command-line utility to manage data and execute workflows on multi-tiered storage systems. It uses a Virtual Data Space (VDS) as an abstraction of the data in a workflow to hide the complexities of the underlying storage systems from the user, while allowing them to control data management strategies.

2 Installation

MaDaTS requires Python (>= 2.7) and pip (>= 9.0) for installation.

To install MaDaTS, run the following commands:

```
$ pip install -r requirements.txt
$ python setup.py install
```

3 Using MaDaTS

3.1 Defining the Storage Hierarchy

The hierarchy of a multi-tiered storage system is defined as a YAML file in MaDaTS. The configuration file in config/storage.yaml contains the list of different storage tiers and the storage-specific properties. You can specify a name of the storage system using the key 'system'. The corresponding value will contain the different tiers and their respective properties.

The example below shows the multi-tiered storage system at NERSC, with four storage tiers, namely burstbuffer, scratch, project and home. MaDaTS will dynamically manage data between these storage tiers for a given workflow.

```
system: nersc
nersc:
                burstbuffer:
                    mount: /mountpts/burstbuffer
                    persist: None
                    interface: posix
                    bandwidth: 1600
                scratch:
                    mount: /scratch/scratchdirs/cscratch1
                    persist: ShortTerm
                    interface: posix
                    bandwidth: 700
                project:
                    mount: /project/projectdirs/test
                    persist: LongTerm
                    interface: posix
                    bandwidth: 40
                home:
                    mount: /home
                    persist: ShortTerm
                    interface: posix
                    bandwidth: 10
```

3.2 Describing a Workflow for MaDaTS

The workflow specification in MaDaTS requires you to describe the task and it's associated properties in YAML format.

The workflow specification is a dictionary consisting of workflow tasks and their associated properties. The keys in the dictionary are identifiers for the tasks. Each task has a set of properties associated with it. Mentioned below are the elements that are used to define the properties of a task.

1. **command**: describes the command to be executed as part of the workflow task.
   [Type = string]
2. **params** : lists the set of parameters of the workflow task
   [Type = list]
3. **vin** : input files to the workflow task to be mapped onto VDS. The input files must be parameters of the command.
   [Type = list]
4. **vout** : output files of the workflow task to be mapped onto VDS. The output files must be parameters of the command.
   [Type = list]
5. **scheduler**: the name of the batch scheduler through which the tasks will be executed.
   scheduler: slurm, uses the Slurm scheduler to manage the task resources.
   scheduler: pbs, uses the PBS scheduler to manage the resources.
   [Type = string]
6. **scheduler_opts**: specifies the different scheduler options to manage the task. These options are defined in the scheduler's config file. Slurm and PBS has the configuration options defined in config/slurm.cfg and config/pbs.cfg respectively.
   [Type = list of dictionaries]

The example below shows an example workflow description in YAML as given to the MaDaTS command-line tool. The different stages of the workflow are divided into separate tasks, where each task has a set of key-value pairs that describe the particular task and its runtime requirements.

```
task1:
        command: python      # executable command
        params:              # list of parameters to the command
          - test1.py
          - -i
          - file1
          - -o
          - int1
        vin:                 # input files to the task
          - file1
        vout:                # output files of the task
          - int1
```

```
    scheduler: slurm     # scheduler to manage the task (slurm/pbs/none)
    scheduler_opts:      # scheduler options used to run the task
      nodes: 4
      walltime: 00:30:00
      partition: regular

  task2:
    command: python
    params:
      - test2.py
      - int1
      - out2
    vin:
      - int1
    vout:
      - out2
    scheduler: slurm
    scheduler_opts:
      nodes: 2
      walltime: 00:15:00
      partition: regular
```

3.3 Python Program using MaDaTS

You can also use the Python library to build and execute a workflow using MaDaTS. You need to import the `madats` module into your Python program.

```
import madats
```

MaDaTS uses the virtual data space (VDS) to manage data across multiple storage tiers. You need to map filesystem objects onto virtual data objects in VDS using the library functions.

```
# Create a Virtual Data Space (VDS)
vds = madats.VirtualDataSpace()
# Create Virtual Data Object
vdo = madats.VirtualDataObject('/scratch/scratchdirs/cscratch1/file1')
```

You can then create tasks using the command to be executed, and associate the tasks to the virtual data objects.

```
# Create a Task
task = madats.Task(command='cat')
task.params = [vdo]
# Associate tasks to virtual data objects
vdo.consumers = [task]
```

You add these virtual data objects to the VDS.

```
# Add the virtual data object to the VDS
vds.add(vdo)
```

Finally, once all the virtual data objects are added to the VDS, you can request MaDaTS to manage the data and workflow.

```
# Manage data and workflow execution through MaDaTS
madats.manage(vds)
```

The details of the MaDaTS API is described in Section 5.

3.3.1 Example

The example below shows a MaDaTS program, consisting of two tasks. The first task reads two files `in1` and `in2`, and writes the data into `inout1`.

The second task reads the file `inout1` and prints the output to `stdout`.

All the files are kept in `scratch` storage in the program. However during execution, MaDaTS may decide to copy/move some or all of the files from `scratch` to `burstbuffer` (assuming the storage hierarchy described earlier in config/storage.yaml) based on the data management policy. The program uses `STORAGE_AWARE` policy of MaDaTS. The `non_movable` attribute of a virtual Data Object (`vdo2` in this example) tells MaDaTS to not move the file `in2` to another storage tier.

```
from madats import VirtualDataSpace, VirtualDataObject, Task
from madats import Persistence
from madats import Policy
import os
import madats

def main():
    datadir = '/scratch/scratchdirs/cscratch1'
    # create a VDS
    vds = VirtualDataSpace()
    vds.data_management_policy = Policy.STORAGE_AWARE
    print('Data management policy:
{}'.format(Policy.name(vds.data_management_policy)))

    # create VDOs
    vdo1 = VirtualDataObject(os.path.join(datadir, 'in1'))
    vdo2 = VirtualDataObject(os.path.join(datadir, 'in2'))
    vdo2.non_movable = True
    vdo3 = VirtualDataObject(os.path.join(datadir, 'inout1'))

    # create tasks
    task = Task(command='cat')
    task.params = [vdo1, vdo2, '>', vdo3]
```

```
    task1 = Task(command='cat')
    task1.params = [vdo3]

    # define VDO and task associations
    vdo1.consumers = [task]
    vdo2.consumers = [task]
    vdo3.producers = [task]
    vdo3.consumers = [task1]
    vds.add(vdo1)
    vds.add(vdo2)
    vds.add(vdo3)

    # manage VDS
    madats.manage(vds)

if __name__ == '__main__':
    main()
```

The examples/ directory in the source tree contains several example scripts to build a workflow and manage data using MaDaTS.

4 Execution

You need to setup `MADATS_HOME` to use MaDaTS. In bash, you can set the path as:

```
$ export MADATS_HOME=</path/to/madats/source/directory>
```

4.1 Using the Command-line

To use the command-line tool, run:

```
$ madats –w <workflow-description>
```

For a more detailed set of options, refer below:

madats -w WORKFLOW [-l LANGUAGE] [-m {dag,bin}] [-p {none,wfa,sta}]

The options to the command-line tool are:

```
 -w WORKFLOW, --workflow WORKFLOW
                workflow description file (default: None)
 -l LANGUAGE, --language LANGUAGE
                workflow description language (default: yaml)
 -m {dag,bin}, --mode {dag,bin}
                execution mode (default: dag)
 -p {none,wfa,sta}, --policy {none,wfa,sta}
                data management policy (default: none)
```

The command-line tool takes a workflow description and maps it to a VDS. The policies can be one of none/wfa/sta, where they imply *Policy.NONE*, *Policy.WORKFLOW_AWARE* and *Policy.STORAGE_AWARE* repectively.

4.2 Using the API

To use the Python API, you need to import the MaDaTS module into your Python program as shown in Section 3.3. You run the Python program as:

```
$ python <my_madats_program>.py
```

5 MaDaTS API

The MaDaTS API provides several objects and functions to help users manage their workflows and data on multi-tiered storage hierarchy. The API primarily uses three types of objects: Task, VirtualDataObject and VirtualDataSpace.

5.1 Task

The Task defines an object that can be executed by MaDaTS.

5.1.1 Task()

Creates a task for MaDaTS.

SYNOPSIS

```
Task(command, type)
```

PARAMETERS

command	(string) an executable command
type	(TaskType) type of the task. A task can either be a COMPUTE or DATA. (default: COMPUTE)

RETURN VALUES

Task	A task object that can be executed by MaDaTS

5.1.2 Attributes

A task object has several attributes that define the execution semantics of the executable command.

- **params** : (type: list) list of parameters to the executable command. The parameters can be any basic data type (string, char, int, float) or a virtual data object.
- **scheduler** : (type: Scheduler.<type>) the type of scheduler that manages the resources and workflow tasks. The default scheduler is *Scheduler.None*, which means the tasks are executed on the local machine. *Scheduler.SLURM* and *Scheduler.PBS* set the respective batch schedulers on HPC systems.
- **scheduler_opts** : (type: dictionary) the different options for the selected scheduler. The options specify the batch job configuration for the scheduler like number of nodes, walltime, queue etc.
- **prerun** : (type: list) list of pre-processing commands that are executed prior to executing the executable task.
- **postrun** : (type: list) list of post-processing commands that are executed after the task is executed.

5.2 VirtualDataObject

VirtualDataObect is an abstract data object that represents a directory or a file on the filesystem.

5.2.1 VirtualDataObject()

Creates a virtual data object.

SYNOPSIS

```
VirtualDataObject(datapath)
```

PARAMETERS

datapath	(string) Path to a data directory or file

RETURN VALUES

VirtualDataObject	A virtual data object that represents data directory or file

5.2.2 add_producer()

Adds a task to the virtual data object signifying that the task is one of the producers of the data object.

SYNOPSIS

```
add_producer(task)
```

PARAMETERS

task	(Task) a task that generates the virtual data object

RETURN VALUES

None

5.2.3 add_consumer()

Adds a task to the virtual data object signifying that the task is one of the consumers of the data object.

SYNOPSIS

```
add_consumer(task)
```

PARAMETERS

task	(Task) a task that uses the virtual data object

RETURN VALUES

None

5.2.4 Attributes

A virtual data object also allows users to define certain properties about the data.

- **size** : (type: int) total size of the data in bytes.
- **persist** : (type: boolean) sets the persistence of a data object.
- **replication** : (type: int) defines the replication factor, i.e., the number of copies of the data.
- **non_movable** : (type: boolean) forces the virtual data object to be non-movable, i.e., allows the data to be static on one storage tier independent of the selected data management strategy.

5.3 VirtualDataSpace

The VirtualDataSpace object is a data space for workflows that allows users to manage data and workflow.

5.3.1 VirtualDataSpace()

Creates a virtual data space (VDS). The data management functions in MaDaTS simply manage a VDS.

SYNOPSIS

```
VirtualDataSpace()
```

PARAMETERS

None

RETURN VALUES

VirtualDataSpace	An empty virtual data space.

5.3.2 map()

Maps a datapath to a virtual data object on VDS.

SYNOPSIS

```
map(datapath)
```

PARAMETERS

datapath	(string) Path to a data directory or file

RETURN VALUES

VirtualDataObject	A virtual data object that represents the data directory or file

5.3.3 add()

Adds a virtual data object to the VDS.

SYNOPSIS

```
add(vdo)
```

PARAMETERS

vdo	(VirtualDataObject) A virtual data object that needs to be added to the VDS

RETURN VALUES

None

5.3.4 copy()

Copies a virtual data object into another virtual data object on VDS.

SYNOPSIS

```
copy(vdo, storage_id)
```

PARAMETERS

vdo	(VirtualDataObject) a virtual data object that needs to be copied.
storage_id	(string) Name of the storage tier to which the copied virtual data object belongs.

RETURN VALUES

VirtualDataObject	The copied virtual data object that is on the specified storage tier.

5.3.5 replace()

Replaces a virtual data object with another virtual data object on VDS.

SYNOPSIS

```
replace(old_vdo, new_vdo)
```

PARAMETERS

old_vdo	(VirtualDataObject) The virtual data object that would be replaced.
new_vdo	(VirtualDataObject) The virtual data object that replaces the other virtual data object.

RETURN VALUES

None

5.3.6 delete()

Removes a virtual data object from VDS.

SYNOPSIS

```
delete(vdo)
```

PARAMETERS

vdo	(VirtualDataObject) The virtual data object that would be deleted.

RETURN VALUES

None

5.3.7 Attributes

A virtual data space allows users to select the data management strategies based on which the workflow and data will be managed by MaDaTS.

- **data_management_policy** : (type: Policy) sets up the policy based on which the data will be managed on multiple storage tiers. By default, MaDaTS does not set any default data management policy as defined by *Policy.NONE*. MaDaTS also has two policies that the users can select from. *Policy.WORKFLOW_AWARE* uses the structure of the workflow to optimize data management, and *Policy.STORAGE_AWARE* uses the properties of the underlying storage tiers to optimize the data management.
- **auto_cleanup** : (type: boolean) enables auto cleaning in MaDaTS for optimizing storage space. If this flag is set, MaDaTS removes copies of the data from the storage tiers if they are no more used by any subsequent tasks.

5.4 Data and Workflow Management

MaDaTS provides three interfaces to manage data and workflows on multi-tiered storage systems.

5.4.1 map()

Maps a workflow on to VDS. This transforms a workflow specification defined in terms of tasks and inputs/outputs to a VDS consisting of virtual data objects and tasks as their producers and consumers.

SYNOPSIS

```
map(workflow, lang, policy)
```

PARAMETERS

workflow	(string/Object) Name of a YAML workflow specification file or a dictionary object describing the workflow. The workflow description must follow the specification supported by MaDaTS (described later).
lang	(string) Describes the language of the workflow description. If the description file is written in Yaml, lang='yaml', and if it's a dictionary object then lang='DictObj'.
policy	(Policy) data management policy as defined by MaDaTS.

RETURN VALUES

VirtualDataSpace	A virtual data space containing virtual data objects

5.4.2 manage()

Manages a VDS. Creates separate tasks to manage data and execute workflow based on the data management policies.

SYNOPSIS

```
manage(vds, execute_mode)
```

PARAMETERS

vds	(VirtualDataSpace) A virtual data space consisting of virtual data objects and associated tasks.
execute_mode	(ExecutionMode) Specifies how the MaDaTS manages the execution of tasks. *ExecutionMode.DAG* executes the workflow with data and compute tasks as a graph, and manages the dependencies accordingly. Each task of the workflow is submitted as a single batch job. *ExecutionMode.BIN* combines several tasks into one job and submits the job to the batch scheduler.

RETURN VALUES

None

5.4.3 query()

Queries a VDS. Allows users to retrieve information about the VDS.

SYNOPSIS

```
query(vds, query)
```

PARAMETERS

vds	(VirtualDataSpace) A virtual data space consisting of virtual data objects and associated tasks.
query	(list) Describes a query with comma-separated metrics. The metrics are: ○ *num_vdos*: number of virtual data objects in a VDS ○ *data_tasks*: number of data tasks created for managing data ○ *data_movements*: number of data movements between the storage tiers ○ *preparer_tasks*: number of tasks preparing the data directories prior to moving data ○ *cleanup_tasks*: number of cleanup tasks for removing unused data

RETURN VALUES

Dict Object	A dictionary containing the queried metrics and associated values

5.4.4 validate()

Validates the completeness of a VDS. It checks if all the virtual data objects referenced by the tasks are added to the VDS.

SYNOPSIS

```
validate(vds)
```

PARAMETERS

vds	(VirtualDataSpace) A virtual data space consisting of virtual data objects and associated tasks.

RETURN VALUES

boolean	True, if all the virtual data objects associated with the tasks are added to the VDS. Else, False.