Modules

da sys

Classes

da.sim.DistProcess(builtins.object)

Node

da.sim.NodeProcess(da.sim.DistProcess)

Node_

class **Node**(da.sim.DistProcess)

```
Abstract base class for DistAlgo processes.

Each instance of this class enbodies the runtime state and activities of a
DistAlgo process in a distributed system. Each process is uniquely
identified by a `ProcessId` object. Messages exchanged between DistAlgo
processes can be any picklable Python object.

DistAlgo processes can spawn more processes by calling `new`. The process
that called `new` is known as the parent process of the newly spawned
processes. Any DistProcess can send messages to any other DistProcess, given
that it knows the `ProcessId` of the target process. However, only the
parent process can `end` a child process. The terminal is shared between all
processes spawned from that terminal, which includes the stdout, stdin, and
stderr streams.

Concrete subclasses of `DistProcess` must define the methods:

- `setup`: A function that initializes the process-local variables.

- `run`: The entry point of the process. This function defines the
  activities of the process.

Users should not instantiate this class directly, process instances should
be created by calling `new`.
```

Method resolution order:

Node
da.sim.DistProcess
builtins.object

Methods defined here:

__init__(self, procimpl, forwarder, **props)

```
Initialize self.  See help(type(self)) for accurate signature.
```

addToRoutingTable(self, X)

```
Identity map the routing table
```

getClosestNode(self, X)

```
Given a node X, determine the closest node based on the proximity metric
```

getCoordinates(self)

```
Map the nodes in a 2D grid to determine the relative distance between
the nodes
```

getHopsCount(self, A, X)

```
This function is used to measure routing performance and returns the
minimum number of hops between any two node A and X
```

getNodeKey(self, ipAddress)

```
Get a 128-bit identifier by hashing the ip address
```

getRandomIPAddress(self)

Repair the leaf set for node X to account for its deletion

repairNeighborSet(self, X)
Repair the neighborhood set for node X to account for its deletion

repairRoutingTable(self, X)
Repair the routing table entry for finding replacement node for node X

route(self, msg, keyid)
Causes Pastry to route the given message to the node with nodeId
numerically closest to the key, among all live Pastry nodes

run(self)
Entry point for the DistAlgo process.

This is the starting point of execution for user code.

sendJoinMessage(self, X)
Sends a join message on node arrival to find the nearest existing node Z
whose id is numerically closest to X
Returns:
The set of nodes encountered on the path from A to Z

sendState(self, X)
Sending state to node X

setup(self, id, **rest_475)
Initialization routine for the DistAlgo process.

Should be overridden by child classes to initialize process states.

shl(self, A, B)
Returns:
The length of the prefix shared among A and B

updateLeafSet(self, Z, X)
Initialize X's leaf set with Z's leaf set

updateNeighbourSet(self, A, X)
Initialize X's neighborhood set with A's neighborhood set

updateRoutingTable(self, routingTable, leafSet, neighborhoodSet)
Update the routing table of the node X based on the closest nodes
encountered present in the routePath

Methods inherited from da.sim.DistProcess:

__repr__(self)
Return repr(self).

__str__ = __repr__(self)
Return repr(self).

debug(self, *message, sep=' ')
Prints debugging output to the process log.

This is the same as `output` except the message is logged at the
'USRDBG' level.

end(self, target, exit_code=1)
Terminate the child processes specified by `target`.

`target` can be a process id or a set of process ids, all of which must
be a child process of this process.

error(self, *message, sep=' ')
Prints error message to the process log.

This is the same as `output` except the message is logged at the
'USRERR' level.

exit(self, code=0)
Terminates the current process.

handlers will no longer run.

incr_logical_clock(self)
Increments the logical clock.

For Lamport's clock, this increases the clock value by 1.

logical_clock(self)
Returns the current value of the logical clock.

nameof(self, pid)
Returns the process name of `pid`, if any.

new(self, pcls, args=None, num=None, at=None, method=None, daemon=False, **props)
Creates new DistAlgo processes.

`pcls` specifies the DistAlgo process class. Optional argument `args` is a list of arguments that is used to call the `setup` method of the child processes. Optional argument `num` specifies the number of processes to create on each node. Optional argument `at` specifies the node or nodes on which the new processes are to be created. If `num` is not specified then it defaults to one process. If `at` is not specified then it defaults to the same node as the current process. Optional argument `method` specifies the type of implementation used to run the new process(es), and can be one of 'process', in which case the new processes will be run inside operating system processes, or 'thread' in which case the processes will be run inside operating system threads. If method is not specified then its default value is taken from the '--default_proc_impl' command line option.

If neither `num` nor `at` is specified, then `new` will return the process id of child process if successful, or None otherwise. If either `num` or `at` is specified, then `new` will return a set containing the process ids of the processes that was successfully created.

nodeof(self, pid)
Returns the process id of `pid`'s node process.

output(self, *message, sep=' ', level=21)
Prints arguments to the process log.

Optional argument 'level' is a positive integer that specifies the logging level of the message, defaults to 'logging.INFO'(20). Refer to [https://docs.python.org/3/library/logging.html#levels] for a list of predefined logging levels.

When the level of the message is equal to or higher than the configured level of a log handler, the message is logged to that handler; otherwise, it is ignored. DistAlgo processes are automatically configured with two log handlers:, one logs to the console, the other to a log file; the handlers' logging levels are controlled by command line parameters.

parent(self)
Returns the parent process id of the current process.

The "parent process" is the process that called `new` to create this process.

resolve(self, name)
Returns the process id associated with `name`.

send(self, message, to, channel=None, **rest)
Send a DistAlgo message.

`message` can be any pickle-able Python object. `to` can be a process id or a set of process ids.

work(self)
Waste some random amount of time.

This suspends execution of the process for a period of 0-2 seconds.

Class methods inherited from da.sim.DistProcess:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Data and other attributes inherited from da.sim.DistProcess:

AckCommands = [<Command.NewAck: 16>, <Command.EndAck: 15>, <Command.StartAck: 11>, <Command.SetupAck: 12>, <Command.ResolveAck: 17>, <Command.RPCReply: 31>]

class **Node_**(da.sim.NodeProcess)

```
Abstract base class for DistAlgo processes.

Each instance of this class enbodies the runtime state and activities of a
DistAlgo process in a distributed system. Each process is uniquely
identified by a `ProcessId` object. Messages exchanged between DistAlgo
processes can be any picklable Python object.

DistAlgo processes can spawn more processes by calling `new`. The process
that called `new` is known as the parent process of the newly spawned
processes. Any DistProcess can send messages to any other DistProcess, given
that it knows the `ProcessId` of the target process. However, only the
parent process can `end` a child process. The terminal is shared between all
processes spawned from that terminal, which includes the stdout, stdin, and
stderr streams.

Concrete subclasses of `DistProcess` must define the methods:

- `setup`: A function that initializes the process-local variables.

- `run`: The entry point of the process. This function defines the
  activities of the process.

Users should not instantiate this class directly, process instances should
be created by calling `new`.
```

Method resolution order:

Node_
da.sim.NodeProcess
da.sim.DistProcess
builtins.object

Methods defined here:

__init__(self, procimpl, forwarder, **props)
Initialize self. See help(type(self)) for accurate signature.

run(self)
Entry point for the DistAlgo process.

This is the starting point of execution for user code.

Methods inherited from da.sim.NodeProcess:

bootstrap(self)

Data and other attributes inherited from da.sim.NodeProcess:

AckCommands = [<Command.NewAck: 16>, <Command.EndAck: 15>, <Command.StartAck: 11>, <Command.SetupAck: 12>, <Command.ResolveAck: 17>, <Command.RPCReply: 31>, <Command.NodeAck: 18>]

Methods inherited from da.sim.DistProcess:

__repr__(self)
Return repr(self).

```
    This is the same as `output` except the message is logged at the
    'USRDBG' level.
```

end(self, target, exit_code=1)
```
    Terminate the child processes specified by `target`.

    `target` can be a process id or a set of process ids, all of which must
    be a child process of this process.
```

error(self, *message, sep=' ')
```
    Prints error message to the process log.

    This is the same as `output` except the message is logged at the
    'USRERR' level.
```

exit(self, code=0)
```
    Terminates the current process.

    `code` specifies the exit code.
```

hanged(self)
```
    Hangs the current process.

    When a process enters the 'hanged' state, its main logic and all message
    handlers will no longer run.
```

incr_logical_clock(self)
```
    Increments the logical clock.

    For Lamport's clock, this increases the clock value by 1.
```

logical_clock(self)
```
    Returns the current value of the logical clock.
```

nameof(self, pid)
```
    Returns the process name of `pid`, if any.
```

new(self, pcls, args=None, num=None, at=None, method=None, daemon=False, **props)
```
    Creates new DistAlgo processes.

    `pcls` specifies the DistAlgo process class. Optional argument `args` is
    a list of arguments that is used to call the `setup` method of the child
    processes. Optional argument `num` specifies the number of processes to
    create on each node. Optional argument `at` specifies the node or nodes
    on which the new processes are to be created. If `num` is not specified
    then it defaults to one process. If `at` is not specified then it
    defaults to the same node as the current process. Optional argument
    `method` specifies the type of implementation used to run the new
    process(es), and can be one of 'process', in which case the new
    processes will be run inside operating system processes, or 'thread' in
    which case the processes will be run inside operating system threads. If
    method is not specified then its default value is taken from the
    '--default_proc_impl' command line option.

    If neither `num` nor `at` is specified, then `new` will return the
    process id of child process if successful, or None otherwise. If either
    `num` or `at` is specified, then `new` will return a set containing the
    process ids of the processes that was successfully created.
```

nodeof(self, pid)
```
    Returns the process id of `pid`'s node process.
```

output(self, *message, sep=' ', level=21)
```
    Prints arguments to the process log.

    Optional argument 'level' is a positive integer that specifies the
    logging level of the message, defaults to 'logging.INFO'(20). Refer to
    [https://docs.python.org/3/library/logging.html#levels] for a list of
    predefined logging levels.

    When the level of the message is equal to or higher than the
    configured level of a log handler, the message is logged to that
    handler; otherwise, it is ignored. DistAlgo processes are
    automatically configured with two log handlers:, one logs to the
    console, the other to a log file; the handlers' logging levels are
    controlled by command line parameters.
```

resolve(self, name)
Returns the process id associated with `name`.

send(self, message, to, channel=None, **rest)
Send a DistAlgo message.

`message` can be any pickle-able Python object. `to` can be a process id
or a set of process ids.

setup(self, **rest)
Initialization routine for the DistAlgo process.

Should be overridden by child classes to initialize process states.

work(self)
Waste some random amount of time.

This suspends execution of the process for a period of 0-2 seconds.

Class methods inherited from da.sim.DistProcess:

get_config(key, default=None) from builtins.type
Returns the configuration value for specified 'key'.

Data descriptors inherited from da.sim.DistProcess:

__dict__
dictionary for instance variables (if defined)

__weakref__
list of weak references to the object (if defined)

Data

PatternExpr_365 = (='forward',message,keyid)
PatternExpr_405 = (routingTable,leafSet,neighborhoodSet)

FLOW CHART FOR PASTRY CORE ROUTING ALGORITHM