CS_FedSim's documentation!

Contents

class node.**LowNode**(*name: str, mobility_mode: bool = False*)

A class that represents the module of a low power node.

...

Attributes: **name : *str***

The node of the node to identify it.

mobility_mode : *bool, optional*

Indicates whether the node is mobile or stationary.

class node.**MidNode**(*name: str, mobility_mode: bool = False*)

A class that represents the module of a medium power node.

...

Attributes: **name : *str***

The node of the node to identify it.

mobility_mode : *bool, optional*

Indicates whether the node is mobile or stationary.

class node.**PowNode**(*name: str, mobility_mode: bool = False*)

A class that represents the module of a powerful node.

...

Attributes: **name : *str***

The node of the node to identify it.

mobility_mode : *bool, optional*

Indicates whether the node is mobile or stationary.

class clientSelection.**ClientSelection**(*nodes: list, debug_mode: bool = False*)

A class that represents the basis of the client selection module.

...

Attributes: **nodes : *list***

The list of all nodes in the environment.

debug_mode : *bool*

Indicates if the debug mode is enabled or not.

Methods

get_nodes():	Return the list of all nodes.
get_debug_mode():	Return if the debug_mode is enabled.

get_debug_mode() → bool

Return if the debug_mode is enabled.

Returns: **debug_mode (bool): Indicates if the debug mode is enabled or not.**

Examples

```
>>> clientSelection.get_debug_mode()
```

get_nodes() → list

Return the list of the nodes.

Parameters:

Returns: **nodes (list): the list of the nodes.**

Examples

```
>>> clientSelection.get_nodes()
```

class clientSelection.RandomClientSelection(*nodes: list*, *K: float = 0.1*, *debug_mode: bool = False*)

A class that inherits the client selection module, which selects clients randomly.

...

Attributes:

nodes : *list*

The list of all nodes in the environment.

K : *float*

the percentage of the selection.

debug_mode : *bool*

Indicates if the debug mode is enabled or not.

Methods

random_client_selection():	Returns a randomly selected list of clients with a percentage K.

random_client_selection() → list

Return the list of selected nodes randomlu.

Parameters:

Returns: **selected_clients (list): the list of the nodes.**

Examples

```
>>> randomClientSelection.random_client_selection()
```

class clientSelection.**ResourceClientSelection**(*nodes: list*, *K: float = 0.1*, *debug_mode: bool = False*)

A class that inherits the client selection module, which selects clients according to the strength of their resources.

...

Attributes:

nodes : *list*

The list of all nodes in the environment.

K : *float*

the percentage of the selection.

debug_mode : *bool*

Indicates if the debug mode is enabled or not.

Methods

resource_client_selection():	Returns a list of clients selected according to their power ranking.

resource_client_selection() → list

Return the list of the selected nodes according to their power.

Parameters:

Returns: **selected_clients (list): the list of the nodes.**

Examples

```
>>> resourceClientSelection.resource_client_selection()
```

class consumptionModel.**CPUModel**(*node: node.Node.Node*)

A class that represents the module of the processor consumption model of the nodes.

...

Attributes:

node : *Node*

The node assigned to this model.

Methods

get_node(): Node	Return the node.
set_node():	Assign this consumption model to a node.
check_cpu():	Check if the node has reached the maximum CPU consumption level.
update_cpu():	Update the CPU consumption percentage of the node.

check_cpu() → bool

Check if the node has reached the maximum CPU consumption level.

Parameters:

Returns: **status (bool): If the maximum level of consumption is reached or not.**

Examples

```
>>> cpuModel.check_cpu()
```

get_node() → node.Node.Node

Return the instance of the node.

Parameters:

Returns: **nodes (Node): The node's instance.**

Examples

```
>>> cpuModel.get_node()
```

set_node(*node: node.Node.Node*)

Assign this consumption model to a node.

Parameters: **node: Node**

the node's instance.

Examples

```
>>> cpuModel.set_node(node1)
```

update_cpu(*cpu_usage*)

Update the CPU consumption percentage of the node.

Parameters: **cpu_usage: float**

the new level of processor consumption.

Examples

```
>>> cpuModel.update_cpu(65)
```

class consumptionModel.EnergyModel(*node: node.Node.Node*)

A class that represents the module of the energy consumption model of the nodes.

...

Attributes: **node : *Node***

The node assigned to this model.

Methods

get_node(): Node	Return the node.
set_node():	Assign this consumption model to a node.
consume_energy():	Consume a certain level of energy from the node according to its category.
check_battery():	Update the CPU consumption percentage of the node.

check_battery()

Check that the node 's battery is not depleted.

Parameters:

Examples

```
>>> cpuModel.check_battery()
```

consume_energy() → float

Consume a certain level of energy from the node according to its category.

Parameters:

Returns: **new_energy (float): The new energy level of the node.**

Examples

```
>>> energyModel.consume_energy()
```

get_node() → node.Node.Node

Return the instance of the node.

Parameters:

Returns: **nodes (Node): The node's instance.**

Examples

```
>>> energyModel.get_node()
```

set_node(*node: node.Node.Node*)

Assign this consumption model to a node.

Parameters: **node: Node**

the node's instance.

Examples

```
>>> energyModel.set_node(node1)
```

class consumptionModel.MemoryModel(*node: node.Node.Node*)

A class that represents the module of the memory consumption model of the nodes.

...

Attributes: **node : *Node***

The node assigned to this model.

Methods

get_node(): Node	Return the node.
set_node():	Assign this consumption model to a node.
check_memory():	Check if the node has reached the maximum Memory consumption level.
update_memory():	Update the Memory consumption percentage of the node.

check_memory() → bool

Check if the node has reached the maximum Memory consumption level.

Parameters:

Returns: **status (bool): If the maximum level of consumption is reached or not.**

Examples

```
>>> memoryModel.check_memory()
```

get_node() → node.Node.Node

Return the instance of the node.

Parameters:

Returns: **nodes (Node): The node's instance.**

Examples

```
>>> memoryModel.get_node()
```

set_node(*node: node.Node.Node*)

Assign this consumption model to a node.

Parameters: **node: Node**

the node's instance.

Examples

```
>>> memoryModel.set_node(node1)
```

update_memory(*memory_usage*)

Update the Memory consumption percentage of the node.

Parameters: **memory_usage: float**

the new level of processor consumption.

Examples

```
>>> memoryModel.update_cpu(57)
```

class consumptionModel.StorageModel(*node: node.Node.Node*)

A class that represents the module of the storage consumption model of the nodes.

...

Attributes: **node : *Node***

The node assigned to this model.

Methods

get_node(): Node	Return the node.
set_node():	Assign this consumption model to a node.
check_storage():	Check if the node has reached the maximum storage consumption level.
add_to_storage():	Add files to the node and thus fill the storage.

add_to_storage(*number_of_mega_bytes: float*) → float

Add files to the node and thus fill the storage.

Parameters: **number_of_mega_bytes: float**

The size of the files added to the storage.

Examples

```
>>> storageModel.add_to_storage(658)
```

check_storage() → bool

Check if the node has reached the maximum Storage consumption level.

Parameters:

Returns: **status (bool): If the maximum level of consumption is reached or not.**

Examples

```
>>> storageModel.check_storage()
```

get_node() → node.Node.Node

Return the instance of the node.

Parameters:

Returns: **nodes (Node): The node's instance.**

Examples

```
>>> storageModel.get_node()
```

set_node(*node: node.Node.Node*)

Assign this consumption model to a node.

Parameters: **node: Node**

the node's instance.

Examples

```
>>> storageModel.set_node(node1)
```

class network.Network(*nodes: list, network_number: int, debug_mode: bool = False*)

A class that represents the module of a network of nodes.

...

Attributes:

nodes : *list*

The list of all nodes in the environment.

network_number : *int*

The network number.

debug_mode : *bool*

Indicates if the debug mode is enabled or not.

Methods

assign_ip_addresses():	Give an ip address to all nodes assigned to this network.
get_network_number():	Return the number of the network.
get_nodes():	Return the list of all nodes.
get_network_leader():	Return the leader of the network.
set_network_leader():	Assign a leader to the network.

`get_network_leader()`

Return the leader of the network.

Parameters:

Returns: **network_leader (Node): the instance of the leader node of the network.**

Examples

```
>>> network.get_network_leader()
```

`get_network_number()`

Return the number of the network.

Parameters:

Returns: **network_number (int): The network number.**

Examples

```
>>> network.get_network_number()
```

`get_nodes() → list`

Return the list of the nodes.

Parameters:

Returns: **nodes (list): the list of the nodes.**

Examples

```
>>> network.get_nodes()
```

set_network_leader(network_leader)

Assign a leader to the network.

Parameters: **network_leader : *Node***

The new node leader for the network.

Returns: **network_leader (Node): the instance of the leader node of the network.**

Examples

```
>>> network.set_network_leader(node1)
```

distribuedLearning.*dist_learning(train_dataset, selected_clients: list, global_model, global_round: int) -> (<class 'float'>, <class 'list'>, <class*

'dict'>, <class 'float'>)

The function of distributed learning of nodes.

Parameters:

train_dataset : *any*

The training dataset used for distributed learning.

selected_clients : *list*

The selected clients.

global_model : *any*

The model global.

global_round : *int*

The current global round.

Returns:

loss_avg (float): the avereage loss.

list_acc (list): the full list of all accuracy.

clients_acc (dict) : *the full list of clients accuracy as dict format.*

energy (float) : *The total energy consumed during distributed learning.*

Examples

```
>>> loss_avg, list_acc, clients_acc, energy =
dist_learning(train_dataset=train_dataset, selected_clients=selected_clients,
global_model=global_model, global_round=epoch)
```

By AICHE Mohamed