ClusterOps API Reference

ClusterOps is a Python library for managing and executing tasks across CPU and GPU resources in a distributed computing environment. It provides functions for resource discovery, task execution, and performance monitoring.

Installation

```bash

$ pip3 install clusterops

```

Table of Contents

1. [CPU Operations](#cpu-operations)
2. [GPU Operations](#gpu-operations)
3. [Utility Functions](#utility-functions)
4. [Resource Monitoring](#resource-monitoring)

CPU Operations

`list_available_cpus()`

Lists all available CPU cores.

Returns

| Type | Description |

|------|-------------|

| `List[int]` | A list of available CPU core indices. |

Raises

| Exception | Description |

|-----------|-------------|

| `RuntimeError` | If no CPUs are found. |

Example

```python

from clusterops import list_available_cpus

available_cpus = list_available_cpus()

print(f"Available CPU cores: {available_cpus}")

```

`execute_on_cpu(cpu_id: int, func: Callable, *args: Any, **kwargs: Any) -> Any`

Executes a callable on a specific CPU.

Parameters

| Name | Type | Description |

|------|------|-------------|

| `cpu_id` | `int` | The CPU core to run the function on. |

| `func` | `Callable` | The function to be executed. |

| `*args` | `Any` | Arguments for the callable. |

| `**kwargs` | `Any` | Keyword arguments for the callable. |

Returns

| Type | Description |

|------|-------------|

| `Any` | The result of the function execution. |

Raises

| Exception | Description |

|-----------|-------------|

| `ValueError` | If the CPU core specified is invalid. |

| `RuntimeError` | If there is an error executing the function on the CPU. |

Example

```python
from clusterops import execute_on_cpu

def sample_task(n: int) -> int:
    return n * n

result = execute_on_cpu(0, sample_task, 10)
print(f"Result of sample task on CPU 0: {result}")
```

`execute_with_cpu_cores(core_count: int, func: Callable, *args: Any, **kwargs: Any) -> Any`

Executes a callable using a specified number of CPU cores.

Parameters

Name	Type	Description
`core_count`	`int`	The number of CPU cores to run the function on.
`func`	`Callable`	The function to be executed.
`*args`	`Any`	Arguments for the callable.
`**kwargs`	`Any`	Keyword arguments for the callable.

Returns

Type	Description
`Any`	The result of the function execution.

Raises

Exception	Description
`ValueError`	If the number of CPU cores specified is invalid or exceeds available cores.
`RuntimeError`	If there is an error executing the function on the specified CPU cores.

Example

```python
from clusterops import execute_with_cpu_cores
```

```
def parallel_task(n: int) -> int:
    return sum(range(n))

result = execute_with_cpu_cores(4, parallel_task, 1000000)
print(f"Result of parallel task using 4 CPU cores: {result}")
```

GPU Operations

`list_available_gpus() -> List[str]`

Lists all available GPUs.

Returns

Type	Description
`List[str]`	A list of available GPU names.

Raises

Exception	Description
`RuntimeError`	If no GPUs are found.

Example

```python
```

```
from clusterops import list_available_gpus

available_gpus = list_available_gpus()
print(f"Available GPUs: {available_gpus}")
```

`select_best_gpu() -> Optional[int]`

Selects the GPU with the most free memory.

Returns

Type	Description
`Optional[int]`	The GPU ID of the best available GPU, or None if no GPUs are available.

Example

```python
from clusterops import select_best_gpu

best_gpu = select_best_gpu()
if best_gpu is not None:
    print(f"Best GPU for execution: GPU {best_gpu}")
else:
    print("No GPUs available")
```

`execute_on_gpu(gpu_id: int, func: Callable, *args: Any, **kwargs: Any) -> Any`

Executes a callable on a specific GPU using Ray.

Parameters

Name	Type	Description
`gpu_id`	`int`	The GPU to run the function on.
`func`	`Callable`	The function to be executed.
`*args`	`Any`	Arguments for the callable.
`**kwargs`	`Any`	Keyword arguments for the callable.

Returns

Type	Description
`Any`	The result of the function execution.

Raises

Exception	Description
`ValueError`	If the GPU index is invalid.
`RuntimeError`	If there is an error executing the function on the GPU.

Example

```python
from clusterops import execute_on_gpu
```

```
def gpu_task(n: int) -> int:
    return n ** 2

result = execute_on_gpu(0, gpu_task, 10)
print(f"Result of GPU task on GPU 0: {result}")
```

`execute_on_multiple_gpus(gpu_ids: List[int], func: Callable, all_gpus: bool = False, timeout: float = None, *args: Any, **kwargs: Any) -> List[Any]`

Executes a callable across multiple GPUs using Ray.

Parameters

Name	Type	Description
`gpu_ids`	`List[int]`	The list of GPU IDs to run the function on.
`func`	`Callable`	The function to be executed.
`all_gpus`	`bool`	Whether to use all available GPUs (default: False).
`timeout`	`float`	Timeout for the execution in seconds (default: None).
`*args`	`Any`	Arguments for the callable.
`**kwargs`	`Any`	Keyword arguments for the callable.

Returns

Type	Description

| `List[Any]` | A list of results from the execution on each GPU. |

Raises

| Exception | Description |

|-----------|-------------|

| `ValueError` | If any GPU index is invalid. |

| `RuntimeError` | If there is an error executing the function on the GPUs. |

Example

```python
from clusterops import execute_on_multiple_gpus

def multi_gpu_task(n: int) -> int:
    return n ** 3

results = execute_on_multiple_gpus([0, 1], multi_gpu_task, 5)
print(f"Results of multi-GPU task: {results}")
```

`distributed_execute_on_gpus(gpu_ids: List[int], func: Callable, *args: Any, **kwargs: Any) -> List[Any]`

Executes a callable across multiple GPUs and nodes using Ray's distributed task scheduling.

Parameters

| Name | Type | Description |

|------|------|-------------|

| `gpu_ids` | `List[int]` | The list of GPU IDs across nodes to run the function on. |

| `func` | `Callable` | The function to be executed. |

| `*args` | `Any` | Arguments for the callable. |

| `**kwargs` | `Any` | Keyword arguments for the callable. |

Returns

| Type | Description |

|------|-------------|

| `List[Any]` | A list of results from the execution on each GPU. |

Example

```python
from clusterops import distributed_execute_on_gpus

def distributed_task(n: int) -> int:
    return n ** 4

results = distributed_execute_on_gpus([0, 1, 2, 3], distributed_task, 3)
print(f"Results of distributed GPU task: {results}")
```

Utility Functions

`retry_with_backoff(func: Callable, retries: int = RETRY_COUNT, delay: float = RETRY_DELAY,

*args: Any, **kwargs: Any) -> Any`

Retries a callable function with exponential backoff in case of failure.

Parameters

Name	Type	Description
`func`	`Callable`	The function to execute with retries.
`retries`	`int`	Number of retries (default: RETRY_COUNT from env).
`delay`	`float`	Delay between retries in seconds (default: RETRY_DELAY from env).
`*args`	`Any`	Arguments for the callable.
`**kwargs`	`Any`	Keyword arguments for the callable.

Returns

Type	Description
`Any`	The result of the function execution.

Raises

Exception	Description
`Exception`	After all retries fail.

Example

```python
from clusterops import retry_with_backoff
```

```
def unstable_task():
    # Simulating an unstable task that might fail
    import random
    if random.random() < 0.5:
        raise Exception("Task failed")
    return "Task succeeded"

result = retry_with_backoff(unstable_task, retries=5, delay=1)
print(f"Result of unstable task: {result}")
```

Resource Monitoring

`monitor_resources()`

Continuously monitors CPU and GPU resources and logs alerts when thresholds are crossed.

Example

```python
from clusterops import monitor_resources

# Start monitoring resources
monitor_resources()
```

`profile_execution(func: Callable, *args: Any, **kwargs: Any) -> Any`

Profiles the execution of a task, collecting metrics like execution time and CPU/GPU usage.

Parameters

Name	Type	Description
`func`	`Callable`	The function to profile.
`*args`	`Any`	Arguments for the callable.
`**kwargs`	`Any`	Keyword arguments for the callable.

Returns

Type	Description
`Any`	The result of the function execution along with the collected metrics.

Example

```python
from clusterops import profile_execution

def cpu_intensive_task():
    return sum(i*i for i in range(10000000))

result = profile_execution(cpu_intensive_task)
print(f"Result of profiled task: {result}")
```

This API reference provides a comprehensive overview of the ClusterOps library's main functions, their parameters, return values, and usage examples. It should help users understand and utilize the library effectively for managing and executing tasks across CPU and GPU resources in a distributed computing environment.