DISTRIBUTED MASTER WORKER PARADIGM DOCUMENTATION

This Document describes the Implemented Master-Worker paradigm using **XML-RPC** in Python, here master server coordinates with multiple worker servers to perform distributed tasks. This Master Worker paradigm consists of one Master code, worker code, and multiple client nodes. In this Master Worker paradigm, Master server manages the registration of workers and delegates some data extraction methods to them.

Detail Description of Implementation of Master Worker Paradigm

Master File:

This master file consists of worker registration, data extraction methods, randomly publishing data, server initialization.

Worker Registration:

- The master code maintains a dictionary of worker names (‘workers’) and corresponding XML-RPC server proxies.
- Here ‘register_worker’ function allows the master to dynamically register additional workers

Data Extraction Methods:

- It has data extraction functions like “get_by_location,” “get_by_name,” “get_by_year” are implemented to extract details from workers based on location, name, and a combination of location and year.
- These functions aggregate results from these functions with the help of location, name (or) by year.

Random Data Publishing:

- A worker is selected at random and receives random data created by the “random_publish_data” function.
- This shows how employees' data processing jobs are distributed in a random and dynamic manner.

Worker Node:

- Here it contains two worker nodes, worker 1, worker 2
- Worker 1 node is responsible for queries that are related to the subset starting with (a-m)
- Worker 2 node is responsible for queries that are related to the subset starts with (n-z).

Worker node is responsible for data loading and data retrieval methods.

Data Loading:

- The ‘load_data’ method is used for reading data from a JSON file based on the worker's group ('am' or 'nz').

Data Retrieval methods:

- like the master code, the worker code has functions for obtaining and publishing data records, including “get_by_name,” “get_by_location,” “get_by_year,” “publish_data,” “ get_details_by_year”

Publisher Node:

Data Publishing:

- **This function ‘publish_data’ is designed to publish details to its workers.**
- This function loads data from two JSON files ('data-am.json' and 'data-nz.json') and publishes them to the respective workers.

Client Node:

Client Requests:

- The client uses name, location, and year to filter data requests it sends to the master server.
- To connect with the master server, it makes use of a Server Proxy.

Extra Features:

Dynamic Worker Enrollment:

- This function allows worker nodes to dynamically register with master on startup, rather than hardcoding them.
- When a worker node registers, it notifies its availability, then master assigns work accordingly.

Incremental Data Publish:

- This function helps to publish data records one at a time rather than loading them all at once.
- This function adds data records to data-am.json or data-nz.json with the help of workers accordingly.
- As a result, the worker and master nodes get data in stages, allowing them to conduct queries as new data is delivered.

Failure Monitoring and Query Rerouting:

- The failures of worker nodes can be detected by the master node.
- The master forwards requests to additional worker nodes that are available in case a worker node is unavailable or does not respond within the predetermined timeout.
- Through the automated modification of query distribution in the event of worker failures.

Error handling:

- To provide the client with a consistent experience and to deliver appropriate error messages by not terminating the client and carrying on with the remaining queries, error handling is implemented at every level of the execution.

Master Worker Paradigm:

- This paradigm uses XML-RPC to communicate between the worker and master servers, allowing for remote function invocation.
- The master is responsible for assigning tasks to registered workers dynamically, enabling parallel processing.

- Error handling is used to manage potential issues such as JSON decoding failures and file not found errors.
- The client script demonstrates how clients can communicate with the master server to acquire information.

Usage:

Master Node:

- Run Master node using this command “python3 master.py 23000”
- Then Master node listens for incoming functional queries from clients and sends them to work nodes.

Worker Node:

- As there are two worker nodes for two data sets. We need to run two commands for running.
- Run Worker node 1 using this command “python3 worker.py 23001 am” as worker node 1 works on data records that starts in between (a-m)
- Run Worker node 2using this command “python3 worker.py 23001 nz” as worker node 1 works on data records that starts in between (n-z)

Client Node:

- Run “python client.py 23000” to start the client.
- As we have multiple clients, then run “python {clientfilename}.py” to start the client process.
- This Node contains various kinds of queries by name, location, and year

Publisher:

- Run python publisher.py to open the publisher component.
- The worker nodes get data from the publisher once it has read JSON files.