**Documentation**

**Dataset:**

**Files:** data-am.json, data-nz.json

* The data is organized as key value pairs, the key is name and the value for that key is again organized as key value pairs which includes record id, name, location, year.
* If the user query is by name, then the worker will return all information (record id, name, location, year) under the key and also it takes less time for worker to process the query when compared to query by location and year.
* We maintain two separate JSON data files (i.e., a-m and n-z) based on the first letter of the name, depends on the first letter the worker will store the respective data to the data table.
* Here, workers are designed to handle the data based on the first letter of the name given by the client. Worker 1 will handle a-m and worker 2 will handle n-z.

**Sample Data:**

**"alice": {**

**"record\_id": 2,**

**"name": "alice",**

**"location": "New York City",**

**"year": 2002**

**}**

**Workflow**:

* The client server architecture is used here. For any communication to happen, server should be up and running. And since the master worker paradigm is used here, workers also should be up and running.
* We register master and two workers on the ports and there is no necessity of registering client. The port registration is done through command line arguments., client uses the same command line argument to specify the port where the master is registered to initiate the communication.
* Before taking the query from client, master and workers should be active for processing client’s query. We have used exception handling in such a way that if the workers are not active then empty data will be returned with appropriate message to the client.

**getByName: (**Client is asking for a person living in a particular location**)**

**Client Query:** Get the details of the user with name “**alice**”. Client makes a call to the server i.e., master

**Master:** Master takes the query from the client, then processes it based on the first letter of the query and then forwards it to the appropriate worker. Here first letter of client’s query is ‘a’ so master forwards it to the worker1(a-m) by calling the worker 1 methods, else if the first character is greater than n, then it would call server 2.

**Worker 1:** worker 1 will execute the method called by the master where it checks if the name is present in the dataset and returns the data corresponding to that name. If the data is not present, then data is not present then appropriate error message will be thrown.

**Master:** Master returns data received from worker to the client.

**getByLocation: (**Client is asking for a person living in a particular location**):**

**Client Query:** ‘kansas city’

**Master:** Master takes the query from the client, and then forwards the request to both worker1 and worker2.

**Worker 1:** worker1 will process the request received from master by checking the dataset, filtering it and then returns the names of the persons living at that particular location.

**Worker 2:** worker2 will also process the same as worker1.

**Master:** worker1 and worker2 will return the names of the persons living at the location ‘kansas city’ and then master will merge the results of both workers and sends it to client.

**getByYear: (**Client asking for a person lived in particular location at that particular year**)**

**Client Query:** ‘kansas city’, ‘2012’

**Master:** Master takes the query from the client, and then forwards the request to both worker1 and worker2.

**Worker 1:** worker1 will process the request received from master by checking the dataset, filtering it and then returns the names of the persons living at that location and in particular year.

**Worker 2:** worker2 will process the request received from master by checking the dataset, filtering it and then returns the names of the persons living at that particular location and in particular year.

**Master:** worker1 and worker2 will return the names of the persons living at the location ‘kansas city’ and in the year ‘2012’and then master will merge the results of both workers and sends it to client.

**Additional Functionalities**:

Note: Please look into the folder PA1-additional-functionalities for the code

In this we removed the process of loading only one dataset into a worker and loading all the datasets into all workers and using distributed computing to server the requests from client.

1. **Here, the workers are hardcoded in the master. Instead, let workers register with the master when they start and take the workload**.  
     
   We defined a worker registration function in master which should be called by each new worker that comes into the system at any point of time. The master will save the list of workers available in a list and selects the workers to perform the job in a round robin fashion.
2. **Instead of all data loaded once, publish them one by one by another process (publisher). What happens to the client query then?**  
     
   To send data one by one, we’re introducing publisher service which sends the data to the workers with an added delay. In mean time when the publisher is transmitting the data, client has requested a query by person name. However, that person’s name and data might not be published to the worker then client receives empty data with appropriate message.