report

April 6, 2024

1 Data Graph

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
core_num = [1, 2, 3, 4, 5, 6, 7, 8]
serialize_runtime1 = np.array([1.408030, 1.051130, 1.306007, 1.231852, 1.
 ↪397015, 1.487618, 1.620430, 1.477292])
serialize_runtime2 = np.array([1.130994, 1.088277, 1.17994, 1.419874, 1.392414,␣
 ↪1.450548, 1.659890, 1.328568])
serialize_runtime3 = np.array([1.416620, 1.124020, 1.28522, 1.368292, 1.435665,␣
 ↪1.402581, 1.638739, 1.437643])
serialize_runtime4 = np.array([1.723806, 1.238657, 1.281300, 1.247451, 1.
 ↪524317, 1.448653, 1.794396, 1.466652])
serialize_runtime5 = np.array([1.287954, 1.235631, 1.338389, 1.261094, 1.
 ↪410457, 1.435956, 1.537711, 1.535379])
```

[3]:
```
concurrent_runtime1 = np.array([1.297007, 0.938661, 1.007892, 1.040915, 1.
 ↪169924, 1.193169, 1.201570, 1.413125])
concurrent_runtime2 = np.array([1.396473, 0.903517, 1.008440, 1.001573, 1.
 ↪159631, 1.187628, 1.309417, 1.393773])
concurrent_runtime3 = np.array([1.038664, 1.039637, 0.969992, 1.083132, 1.
 ↪148455, 1.145212, 1.273873, 1.328307])
concurrent_runtime4 = np.array([1.190685, 0.950260, 1.030473, 1.076328, 1.
 ↪075234, 1.182354, 1.272513, 1.334007])
concurrent_runtime5 = np.array([0.981141, 0.970899, 0.962837, 1.025569, 1.
 ↪063953, 1.244475, 1.321956, 1.266431])
```

[4]:
```
serialize_avg_runtime = (serialize_runtime1 + serialize_runtime2 +␣
 ↪serialize_runtime3 + serialize_runtime4 + serialize_runtime5) / 5
concurrent_avg_runtime = (concurrent_runtime1 + concurrent_runtime2 +␣
 ↪concurrent_runtime3 + concurrent_runtime4 + concurrent_runtime5) / 5
```

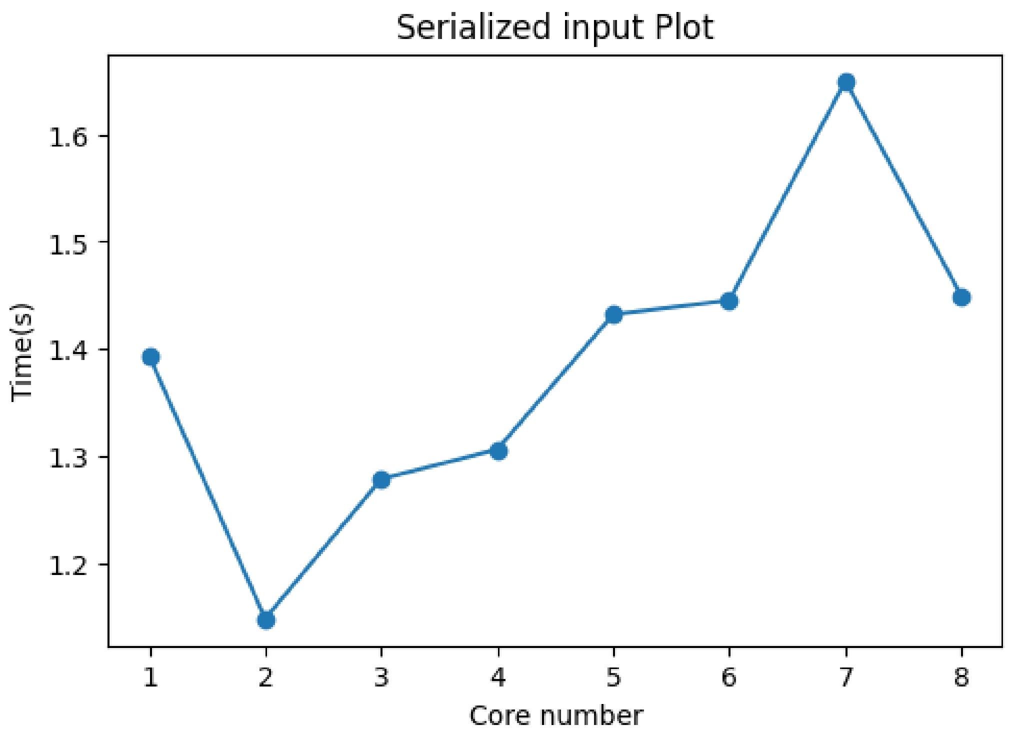
[5]:
```
print(serialize_avg_runtime)
```

```
[1.3934808 1.147543  1.2781712 1.3057126 1.4319736 1.4450712 1.6502332
 1.4491068]
```

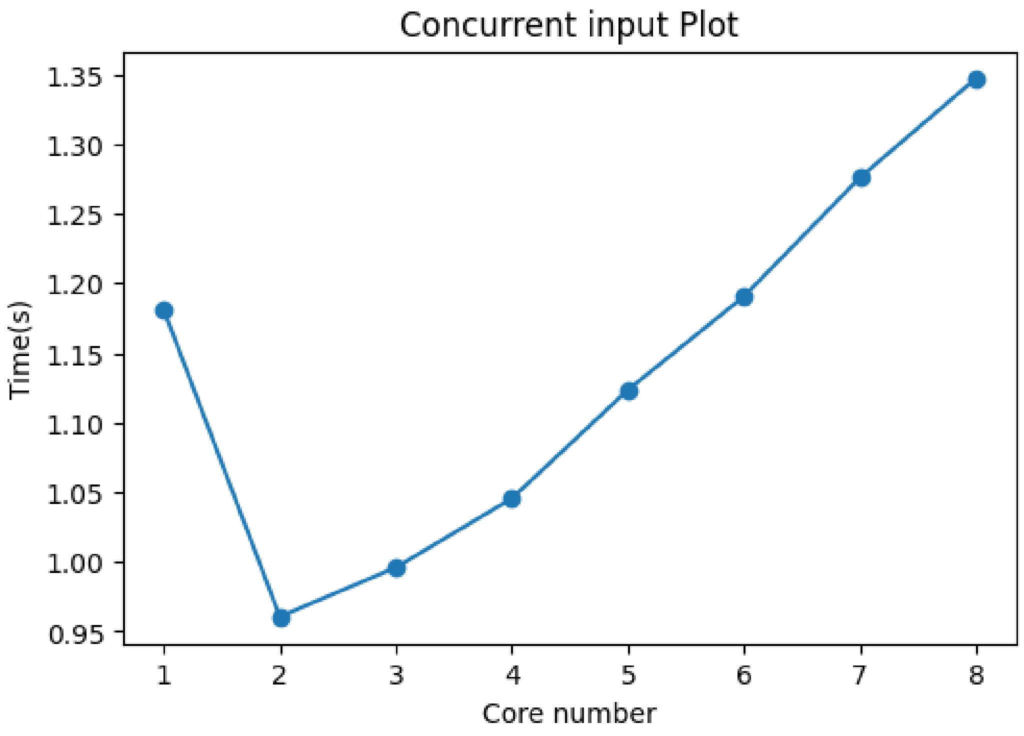
[6]:
```
print(concurrent_avg_runtime)
```

```
[1.180794  0.9605948 0.9959268 1.0455034 1.1234394 1.1905676 1.2758658
 1.3471286]
```

[7]:
```
plt.figure(figsize=(6, 4))
plt.plot(core_num, serialize_avg_runtime, '-o')
plt.title("Serialized input Plot")
plt.xlabel("Core number")
plt.ylabel("Time(s)")
plt.show()
```



[8]:
```
plt.figure(figsize=(6, 4))
plt.plot(core_num, concurrent_avg_runtime, '-o')
plt.title("Concurrent input Plot")
plt.xlabel("Core number")
plt.ylabel("Time(s)")
plt.show()
```



2 Data Analysis

- The first graph is the runtime of serialized input. There is a block between each input, every input needs to wait the previous one to be handled. The fluctuation is caused by overhead of the inner mechanism of python pool.
- The second graph is the runtime of concurrent input. All inputs are handled simultaneously. Time is influenced by overhead of reading and writing database.
- Generally, from the two graphs, we can see that when two cores are used to handle the inputs, it gets the shortest running time. When more cores are used, the cores' computing time doesn't have a obvious increase while the overhead dominates.

3 Design choice

```
exchangemachine_1  | Connection from:  ('127.0.0.1', 41604)
exchangemachine_1  | Connection closed
exchangemachine_1  | Connection from:  ('127.0.0.1', 41616)
exchangemachine_1  | Connection closed
postgres_db_container | 2024-04-06 21:44:18.557 UTC [33] ERROR:  duplicate key value violates u
postgres_db_container | 2024-04-06 21:44:18.557 UTC [33] DETAIL:  Key (id)=(5) already exists.
postgres_db_container | 2024-04-06 21:44:18.557 UTC [33] STATEMENT:  INSERT INTO account (id, b
exchangemachine_1  | Connection from:  ('127.0.0.1', 41632)
exchangemachine_1  | Connection closed
```

```
exchangemachine_1  | Connection from:  ('127.0.0.1', 41644)
exchangemachine_1  | Connection closed
```

Here, we use the primary key error not adding additional detection algorithm is that:

- Relying on the database's built-in constraint checks can simplify application logic. The database itself can ensure data consistency and uniqueness.
- It often more efficient than implementing lock mechanisms at the application layer.