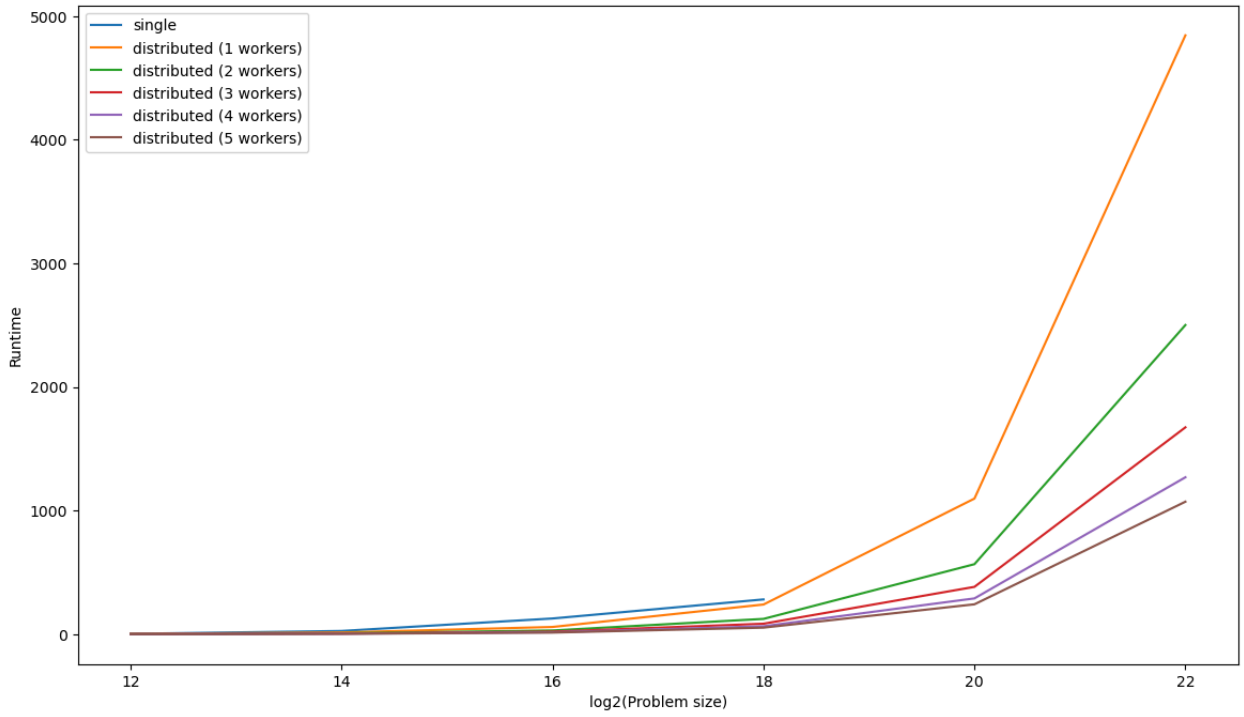
Exercise 2.1

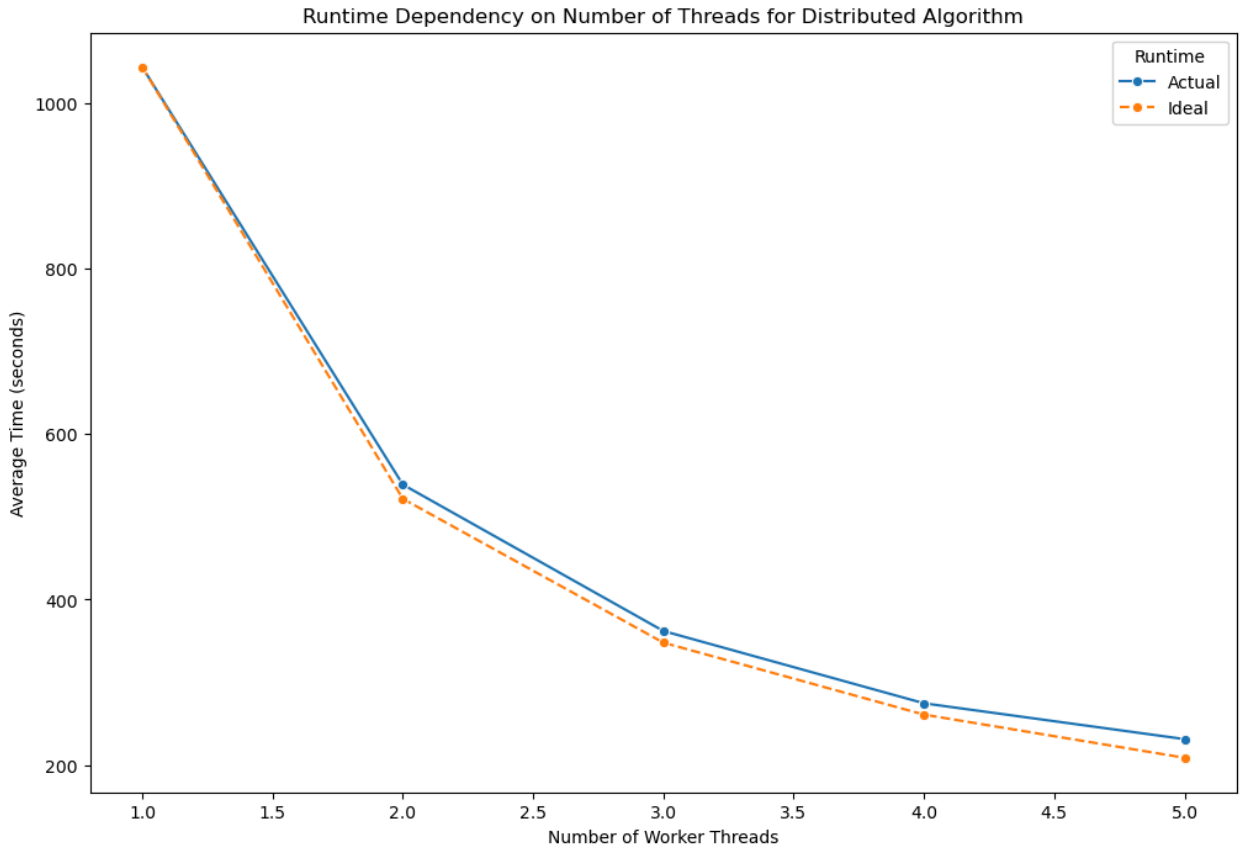
In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import math
import numpy as np
import seaborn as sns


def log2(x):
    return x.apply(lambda x: math.log2(x))

df = pd.read_csv('runtimes.csv')

df['size'] = log2(df['size'])
df_single = df[df['algo'] == 'single']
df_distributed = df[df['algo'] == 'distributed']

workers = df_distributed['workers'].unique()
dfs_workers = {worker: df_distributed[df_distributed['workers'] == worker] for worker in workers}

plt.figure(figsize=(14, 8))

plt.plot(df_single['size'], df_single['time'], label='single')

for worker, df_worker in dfs_workers.items():
    plt.plot(df_worker['size'], df_worker['time'], label=f'distributed ({worker} workers)')

plt.xlabel('log2(Problem size)')
plt.ylabel('Runtime')
plt.legend()
plt.show()
```



In [11]:
```
import numpy as np

average_runtime_per_worker = df_distributed.groupby('workers')['time'].mean().reset_index()

if 1 in average_runtime_per_worker['workers'].values:
    baseline_time = average_runtime_per_worker[average_runtime_per_worker['workers'] == 1]['time'].iloc[0]
    average_runtime_per_worker['ideal_time'] = baseline_time / average_runtime_per_worker['workers']
else:

    min_workers = average_runtime_per_worker['workers'].min()
    baseline_time = average_runtime_per_worker[average_runtime_per_worker['workers'] == min_workers]['time'].iloc[0]
    scale_factor = baseline_time / min_workers
    average_runtime_per_worker['ideal_time'] = scale_factor / average_runtime_per_worker['workers']

plt.figure(figsize=(12, 8))
sns.lineplot(data=average_runtime_per_worker, x='workers', y='time', marker='o', label='Actual')
sns.lineplot(data=average_runtime_per_worker, x='workers', y='ideal_time', marker='o', label='Ideal', linestyle='--')

plt.title('Runtime Dependency on Number of Threads for Distributed Algorithm')
plt.xlabel('Number of Worker Threads')
plt.ylabel('Average Time (seconds)')
```

```
plt.legend(title='Runtime')

plt.show()
```

```
C:\Users\Siddharth\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.

C:\Users\Siddharth\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.

C:\Users\Siddharth\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.

C:\Users\Siddharth\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:

use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
```



Exercise 2.2

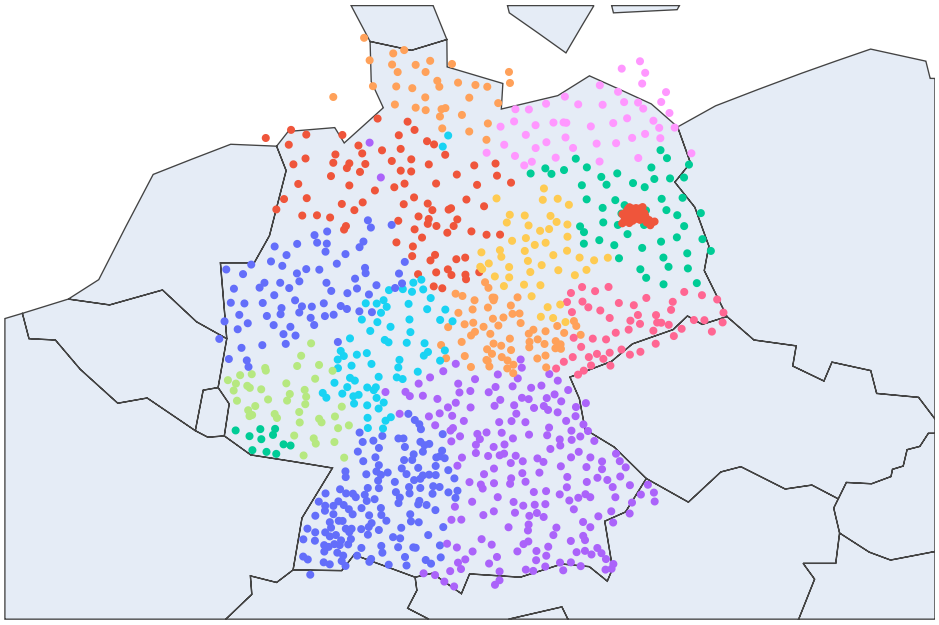
In [3]:
```
import re

with open('zehn_min_rr_Beschreibung_Stationen.txt', 'r', encoding='Windows-1252') as file:
    lines = file.readlines()



pattern = re.compile(r'(\d+)\s+(\d+)\s+(\d+)\s+(\d+)\s+([\d.]+)\s+([\d.]+)\s+([\w\s\/()äöüß-]+)\s+([\w\säöüß-]+)')

parsed_data = []

for line in lines[2:]:
    line = line.strip()
    if line:
        match = pattern.match(line)
        if match:

            parsed_data.append({
                'Stations_id': match.group(1),
                'von_datum': match.group(2),
                'bis_datum': match.group(3),
                'Stationshoehe': match.group(4),
                'geoBreite': match.group(5),
                'geoLaenge': match.group(6),
                'Stationsname': str(match.group(7)).strip(),
                'Bundesland': str(match.group(8)).strip()
            })
```

```
df = pd.DataFrame(parsed_data)
```

In [4]:
```
df.loc[df['Bundesland'] == 'v', 'Bundesland'] = 'Bayern'
df['Bundesland'].unique()
```

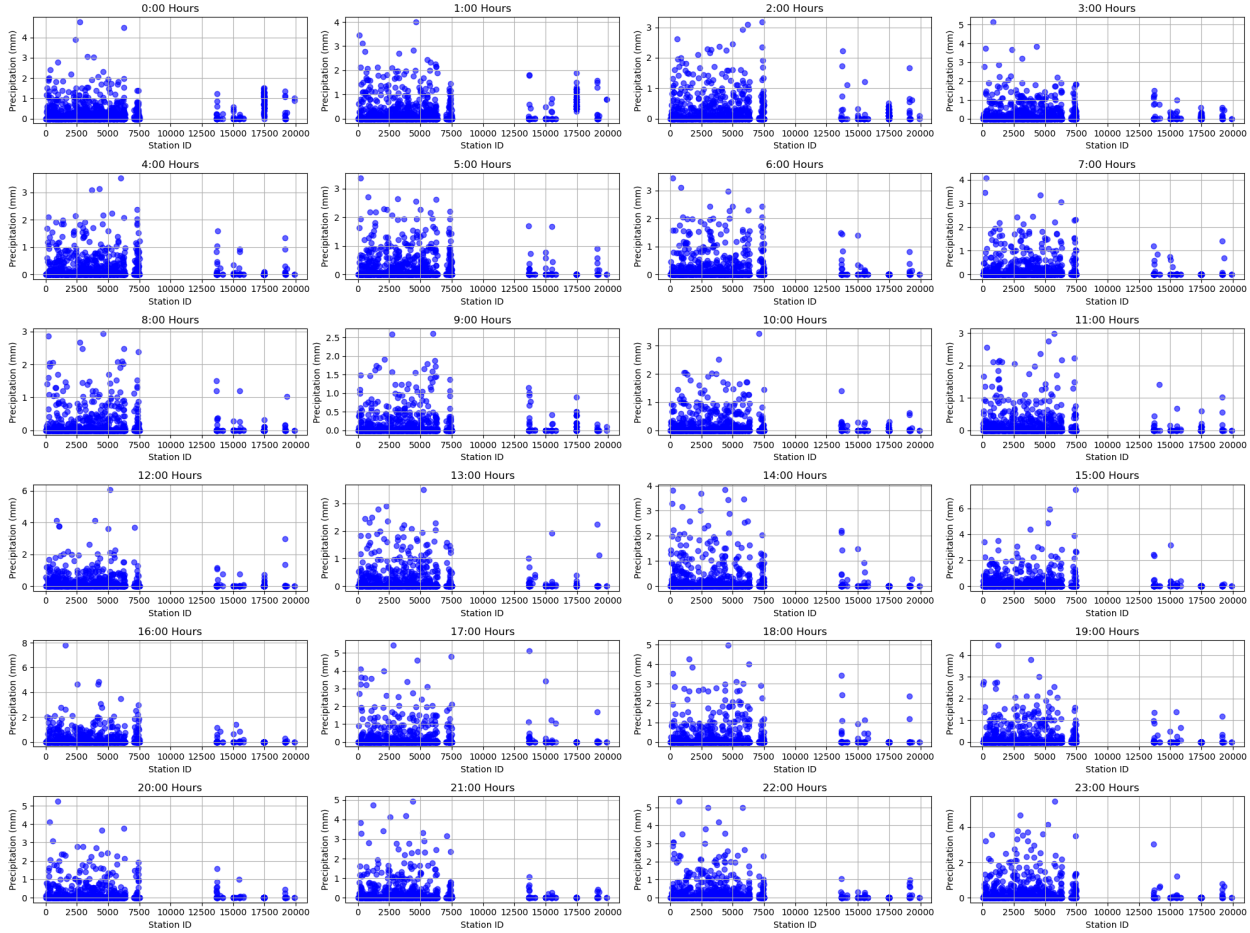
Out[4]:
```
array(['Baden-Württemberg', 'Niedersachsen', 'Brandenburg', 'Bayern',
       'Thüringen', 'Hessen', 'Sachsen', 'Rheinland-Pfalz',
       'Mecklenburg-Vorpommern', 'Sachsen-Anhalt', 'Nordrhein-Westfalen',
       'Berlin', 'Saarland', 'Bremen', 'Schleswig-Holstein', 'Hamburg'],
      dtype=object)
```

In [5]:
```
import plotly.express as px
fig = px.scatter_geo(df, lat="geoBreite", lon="geoLaenge",
                     hover_name="Stationsname", color="Bundesland",
                     scope="europe", projection="natural earth")
fig.update_geos(center=dict(lon=10.4515, lat=51.1657), projection_scale=6)
fig.update_layout(width=1200, height=600)
fig.write_html("weather_stations_map.html")
fig.show()
```



In [6]:
```
data = pd.read_csv('10min_processed.csv')

data['rain'].replace(-999, 0, inplace=True)
data['date'] = pd.to_datetime(data['date'], format='%Y%m%d%H%M')

data['hour'] = data['date'].dt.hour
hourly_precipitation = data.groupby(['stationid', 'hour'])['rain'].sum().reset_index()
```

In [7]:
```
hourly_precipitation = data.groupby(['stationid', 'hour'])['rain'].sum().reset_index()
unique_hours = hourly_precipitation['hour'].unique()
# Create a grid of subplots for each hour using subfigures
fig, axs = plt.subplots(6, 4, figsize=(20, 15)) # Adjust the size as needed
axs = axs.flatten()

for i, hour in enumerate(unique_hours):
    subset = hourly_precipitation[hourly_precipitation['hour'] == hour]
    axs[i].scatter(subset['stationid'], subset['rain'], color='blue', alpha=0.6)
    axs[i].set_title(f'{hour}:00 Hours')
    axs[i].set_xlabel('Station ID')
    axs[i].set_ylabel('Precipitation (mm)')
    axs[i].grid(True)

plt.tight_layout()
plt.show()
```



In [8]:
```
npz = np.load('griddata.npz')
# df= pd.DataFrame.from_dict({item: npz[item] for item in npz.files}, orient='index')

for item in npz.files:
    print(item, npz[item].shape)
```

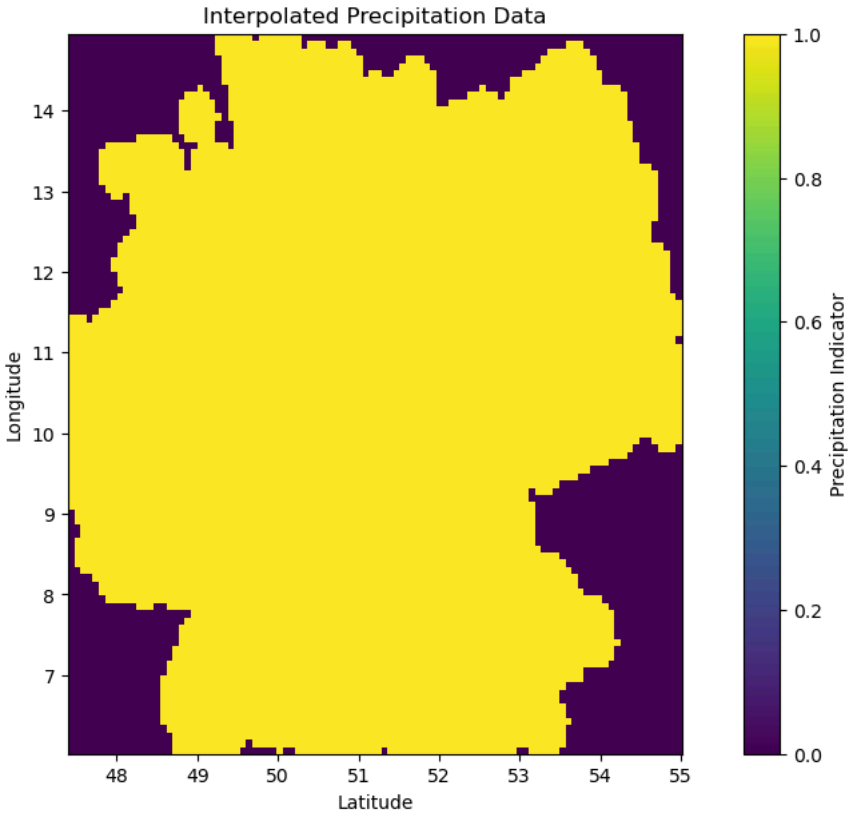
```
geolong (100, 100)
geolat (100, 100)
ind (100, 100)
```

Exercise 2.3

In [9]:
```
import numpy as np
import pandas as pd

# Load the NPZ file
npz = np.load('griddata.npz')

# Create a dictionary to convert to DataFrame
data = np.load('griddata.npz')
geolat = data['geolat']
geolong = data['geolong']
ind = data['ind']
```

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata


data = np.load('griddata.npz')
geolat = data['geolat']
geolong = data['geolong']
ind = data['ind']

grid_x, grid_y = np.meshgrid(np.linspace(geolat.min(), geolat.max(), 100),
                             np.linspace(geolong.min(), geolong.max(), 100))


grid_z = griddata((geolat.flatten(), geolong.flatten()), ind.flatten().astype(float),
                  (grid_x, grid_y), method='linear')

plt.figure(figsize=(12, 7))
plt.imshow(grid_z.T, extent=(geolat.min(), geolat.max(), geolong.min(), geolong.max()),
           origin='lower')
plt.title('Interpolated Precipitation Data')
plt.xlabel('Latitude')
```

```
plt.ylabel('Longitude')
plt.colorbar(label='Precipitation Indicator')
plt.show()
```



In [ ]: