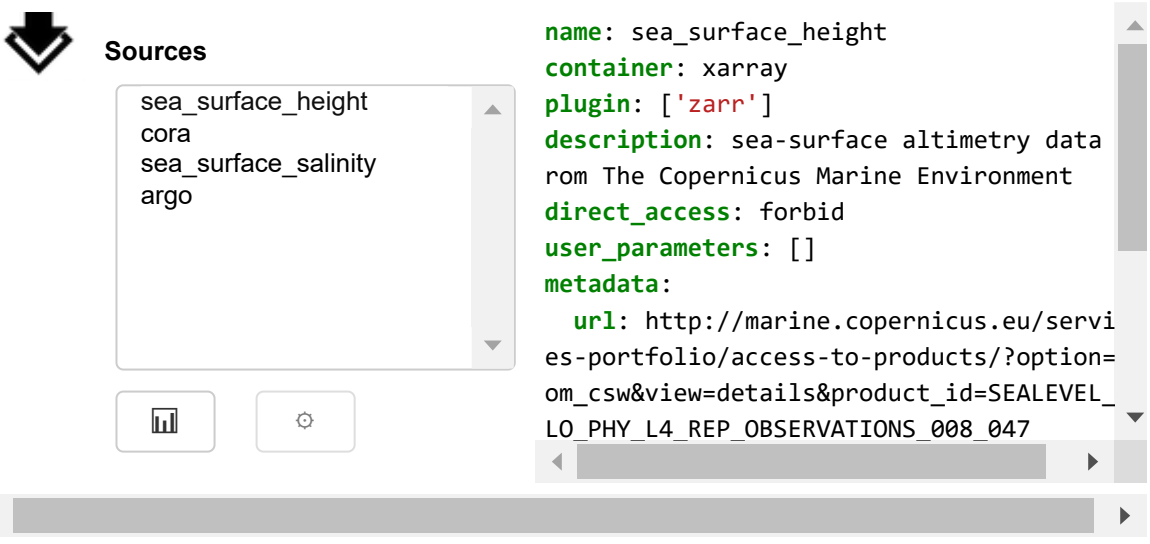
In [1]:

```
from typing import Optional
import datetime
import logging
import re
import os
import dask.dataframe
import dask.distributed
import numpy as np
import pandas as pd
import pyinterp.backends.xarray
import pyinterp.geodetic
import xarray as xr
import time
import intake
import cartopy
cartopy.config['data_dir'] = "/home/datawork-data-terra/conda/cartopy_offline_data"
```

In [2]:

```
intake_cat = intake.open_catalog("/home/datawork-data-terra/eosc-pillar/intake/ocean.yaml")
intake_cat.gui
```

Out[2]:



Local Dask cluster configuration

In [3]:

```
cluster = dask.distributed.LocalCluster(n_workers=4)
client = dask.distributed.Client(cluster)
client
```

Out[3]:

Client
Scheduler: tcp://127.0.0.1:33184
Dashboard: http://127.0.0.1:8787/status (http://127.0.0.1:8787/status)

Cluster
Workers: 4
Cores: 16
Memory: 33.56 GB

Sélection géographique

In [4]:

```
def _select_area(ddf: dask.dataframe.DataFrame, box: pyinterp.geodetic.Box2D):
    """Applies geographic selection to a DataFrame of a partition"""
    return list(
        box.covered_by(ddf.longitude.values, ddf.latitude.values).astype(bool))


def select_area(ddf: dask.dataframe.DataFrame, box: pyinterp.geodetic.Box2D):
    """Applies geographic selection to a DataFrame"""
    return ddf.map_partitions(_select_area, box)
```

In [5]:

```
# Creation of the data selection box.
area = pyinterp.geodetic.Box2D(
    pyinterp.geodetic.Point2D(-57, 10),
    pyinterp.geodetic.Point2D(20, 60))
area
```

Out[5]:

```
((-57, 10), (20, 60))
```

In [6]:

```
ddf = intake_cat.argo(year=[2010], month=range(1, 6)).to_dask()

# Subsetting a specific area.
df = ddf[select_area(ddf, area)].compute()
```

```
/export/home/anaconda/envs/dataterra/lib/python3.7/site-packages/dask/data
frame/io/parquet/arrow.py:680: UserWarning: Filtering with gather_statisti
cs=False. Only partition columns will be filtered correctly.
  "Filtering with gather_statistics=False. "
```

In [7]:

```
len(df)
```

Out[7]:

```
5717
```

Visualizing sea water temperature from Argo dataset

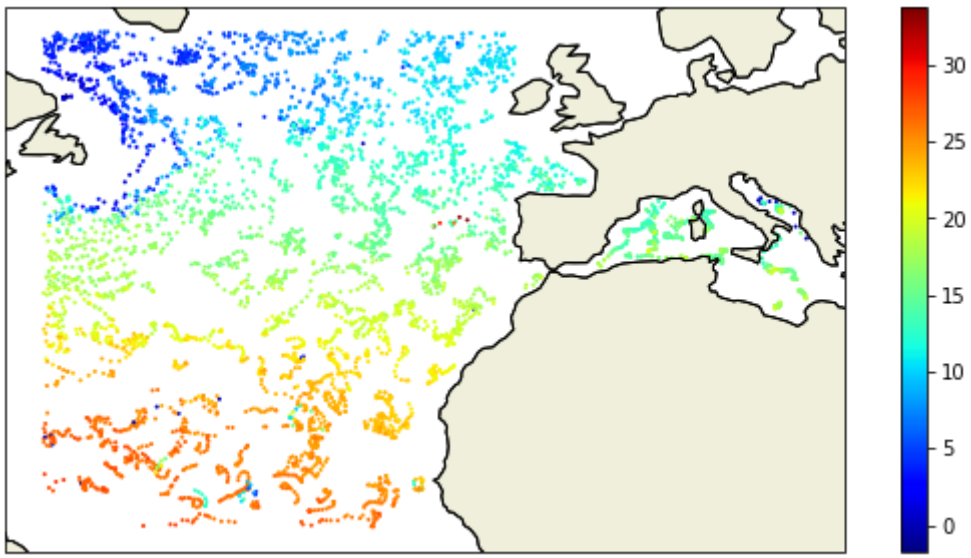
In [8]:

```
# Visualization of the result
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
%matplotlib inline

fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree(central_longitude=0))
sc = ax.scatter(
    df.longitude,
    df.latitude,
    1,
    c=[item[0] for item in df.temp],
    transform=ccrs.PlateCarree(),
    cmap='jet')
ax.coastlines()
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.COASTLINE)
fig.colorbar(sc)
```

Out[8]:

```
<matplotlib.colorbar.Colorbar at 0x7f22b070d150>
```



Computing pressure anomalies from Argo dataset

In [9]:

```
def pressure_anomalies(df):
    """Calculates pressure anomalies"""
    return df.pres - df.pres_adjusted
```

In [10]:

```
# Here only columns containing the longitude and latitude of the floats are
# selected.
df = ddf[['longitude', 'latitude']].compute()
df['anomalies'] = ddf.map_partitions(
    pressure_anomalies, meta=(None, 'f8')).compute()
```

In [11]:

```
# The average anomaly is calculated
df['mean_anomalies'] = df['anomalies'].map(
    lambda series: np.nan if np.all(np.isnan(series)) else np.nanmean(series))
```

In [12]:

```
df
```

Out[12]:

	longitude	latitude	anomalies	mean_anomalies
0	126.232002	-40.058998	[nan, nan, nan, nan, nan, nan]	NaN
1	126.080002	-39.645000	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...	NaN
2	126.553001	-39.356998	[nan, nan]	NaN
3	53.685001	-42.421001	[5.3, 5.299999, 5.300001, 5.299999, 5.300003, ...	5.299996
4	55.987999	-42.901001	[5.6000004, 5.6000004, 5.6000004, 5.5999985, 5...	5.600000
...	...	...	...	...
14289	83.678000	9.917000	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...	NaN
14290	83.575000	9.497000	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...	NaN
14291	85.264000	-0.082000	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...	NaN
14292	86.953000	-0.003000	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...	NaN
14293	85.976000	-0.561000	[nan, nan, nan, nan, nan, nan, nan, nan, nan, ...	NaN

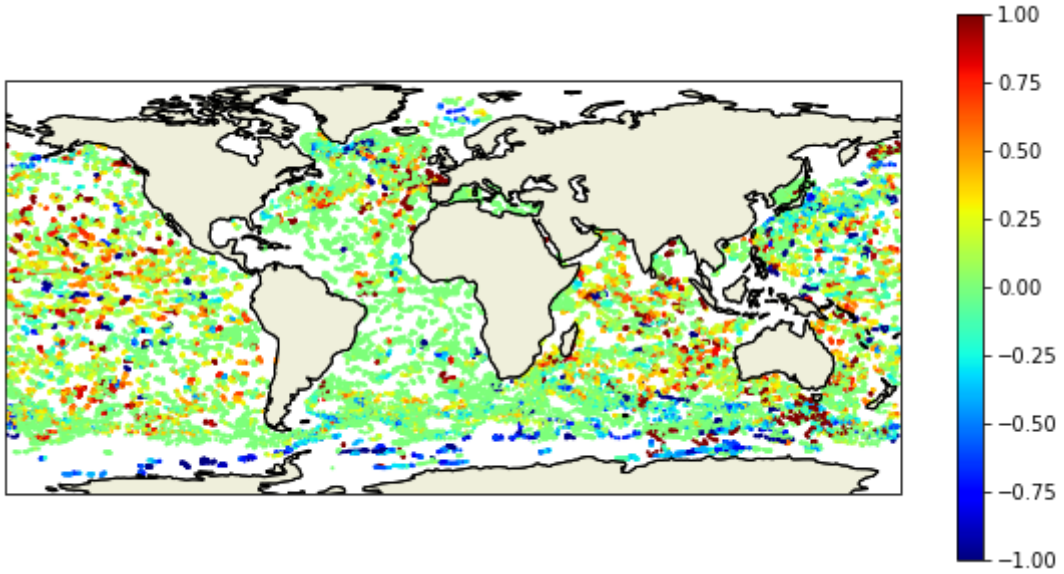
67125 rows × 4 columns

In [13]:

```
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree(central_longitude=0))
sc = ax.scatter(
    df.longitude,
    df.latitude,
    1,
    c=df.mean_anomalies,
    transform=ccrs.PlateCarree(),
    cmap='jet',
    vmin=-1,
    vmax=1)
ax.coastlines()
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.COASTLINE)
fig.colorbar(sc)
```

Out[13]:

```
<matplotlib.colorbar.Colorbar at 0x7f22b16edfd0>
```



SLA interpolation on Argo float positions

In [14]:

```
class GridSeries:
    """Handles a series of grids stored in zarr format. This series is a
    time series."""
    def __init__(self, ds):
        self.ds = ds
        self.series, self.dt = self._load_ts()


    @staticmethod
    def _is_sorted(array):
        indices = np.argsort(array)
        return np.all(indices == np.arange(len(indices)))

    def _load_ts(self):
        """Loading the time series into memory."""
        time = self.ds.time
        assert self._is_sorted(time)

        series = pd.Series(time)
        frequency = set(np.diff(series.values.astype("datetime64[s]")).astype("int64"))
        if len(frequency) != 1:
            raise RuntimeError(
                "Time series does not have a constant step between two "
                f"grids: {frequency} seconds")
        return series, datetime.timedelta(seconds=float(frequency.pop()))

    def load_dataset(self, varname, start, end):
        """Loading the time series into memory for the defined period.

        Args:
            varname (str): Name of the variable to be loaded into memory.
            start (datetime.datetime): Date of the first map to be loaded.
            end (datetime.datetime): Date of the last map to be loaded.

        Return:
            pyinterp.backends.xarray.Grid3D: The interpolator handling the
            interpolation of the grid series.
        """
        if start < self.series.min() or end > self.series.max():
            raise IndexError(
                f"period [{start}, {end}] out of range [{self.series.min()}, "
                f"{self.series.max()}]")
        first = start - self.dt
        last = end + self.dt

#         selected = self.series[(self.series >= first) & (self.series < last)]
        selected = self.series[(self.series >= start) & (self.series < end)]

        data_array = ds[varname].isel(time=selected.index)
        return pyinterp.backends.xarray.Grid3D(data_array)
```

In [15]:

```
def interpolate(df, grid_series, varname):
    """Interpolate a variable 'varname' described by the time series
    'grid_series' for the locations provided in the DataFrame 'df'"""
    if not len(df):
        return np.array([])
    # The DataFrame must be ordered by the time axis
    df = df.set_index("datetime")
    # The time axis is divided into monthly periods
    period_start = df.groupby(df.index.to_period('M'))["sla"].count().index
    periods = []

    end = None

    # Calculates the period required to interpolate the data from the provided
    # time series
    for start, end in zip(period_start, period_start[1:]):
#         print(f"Calculating period : {start} -> {end}")
        start = start.to_timestamp()
        if start < grid_series.df.index[0]:
            start = grid_series.df.index[0]
        end = end.to_timestamp()
        periods.append((start, end))
    if end is None:
        end = period_start[0].to_timestamp()
#         print(f"End is none so it will be : {end}")

#     periods.append((end, df.index[-1] + datetime.timedelta(seconds=3600)))
    periods.append((end, df.index.max() + datetime.timedelta(seconds=3600)))

    # Finally, the data on the different periods identified are interpolated.
    result = []
    for start, end in periods:
        interpolator = grid_series.load_dataset(varname, start, end)
        mask = (df.index >= start) & (df.index < end)
        selected = df.loc[mask, ["longitude", "latitude"]]
        result.append(
            interpolator.trivariate(dict(
                longitude=selected["longitude"].values,
                latitude=selected["latitude"].values,
                time=selected.index.values),
                                    interpolator="inverse_distance_weighting",
                                    num_threads=1))

    return pd.Series(np.hstack(result), df.index)
```

Loading SLA dataset from zarr store

In [16]:

```
ds = intake_cat.sea_surface_height().to_dask()

lon_attrs = ds.longitude.attrs
ds = ds.assign_coords(longitude=(((ds.longitude + 180) % 360) - 180))#.sortby('time')
ds.longitude.attrs = lon_attrs
```

In [17]:

```
# DELETE
ds = ds.drop("crs")
ds
```

Out[17]:

xarray.Dataset

▶ Dimensions: (**latitude**: 720, **longitude**: 1440, **nv**: 2, **time**: 9784)

▼ Coordinates:

latitude	(latitude)	float32	-89.875 -89.62…
longitude	(longitude)	float32	0.125 0.375 0.…
nv	(nv)	int32	0 1
time	(time)	datetime64[ns]	1993-01-01 … …

▼ Data variables:

adt	(time, latitude, longitude)	float64	dask.array<chu…
err	(time, latitude, longitude)	float64	dask.array<chu…
lat_bnds	(time, latitude, nv)	float32	dask.array<chu…
lon_bnds	(time, longitude, nv)	float32	dask.array<chu…
sla	(time, latitude, longitude)	float64	dask.array<chu…
ugos	(time, latitude, longitude)	float64	dask.array<chu…
ugosa	(time, latitude, longitude)	float64	dask.array<chu…
vgos	(time, latitude, longitude)	float64	dask.array<chu…
vgosa	(time, latitude, longitude)	float64	dask.array<chu…

▶ Attributes: (44)

In [18]:

```
grid_series = GridSeries(ds)
```

In [19]:

```
# Calculation of SLA
# sla = ddf.map_partitions(interpolate, grid_series, 'sla', meta=('result', np.float6
4)).compute(scheduler="single-threaded")
sla = ddf.map_partitions(interpolate, grid_series, 'sla', meta=('result', np.float64)).
compute()
```

In [20]:

```
# Generation of a DataFrame containing the float positions and the
# interpolated SLA.
df = ddf[["datetime", "longitude", "latitude"]].compute()
df = df.join(sla, on="datetime")
```

Visualization of the result

In [21]:

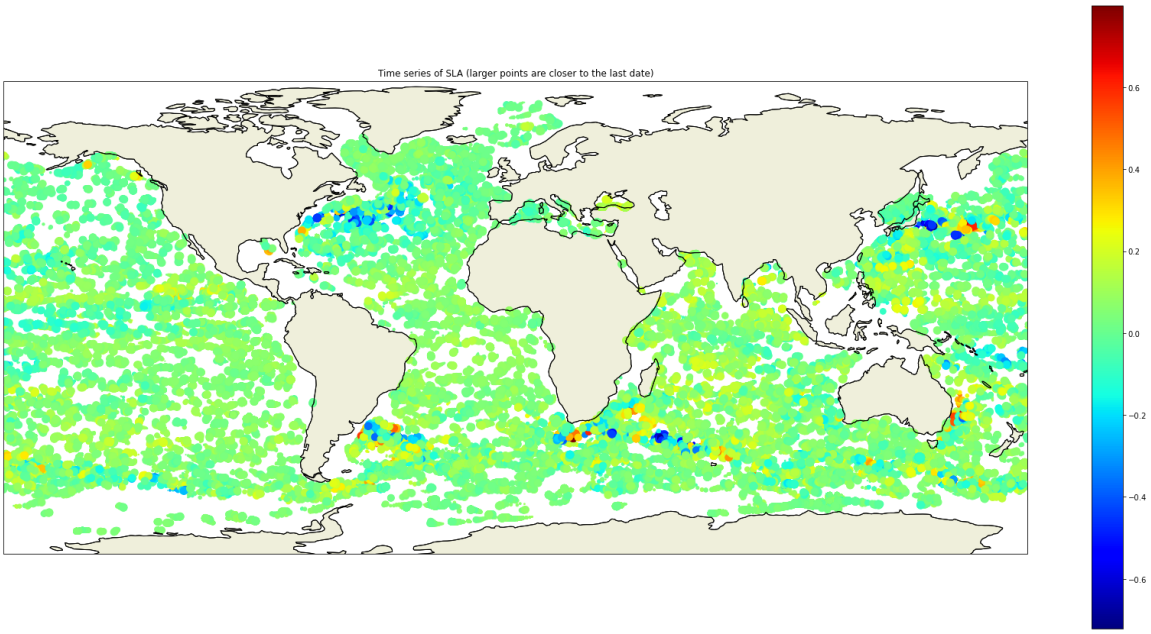
```
first = df.datetime.min()
last = df.datetime.max()
size = (df.datetime - first) / (last-first)
```

In [22]:

```
fig = plt.figure(figsize=(30, 15))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree(central_longitude=0))
sc = ax.scatter(
    df.longitude,
    df.latitude,
    s=size*100,
    c=df.result,
    transform=ccrs.PlateCarree(),
    cmap='jet')
ax.coastlines()
ax.set_title("Time series of SLA "
             "(larger points are closer to the last date)")
ax.add_feature(cfeature.LAND)
ax.add_feature(cfeature.COASTLINE)
# ax.set_extent([80, 100, 13.5, 25], crs=ccrs.PlateCarree())
fig.colorbar(sc)
```

Out[22]:

```
<matplotlib.colorbar.Colorbar at 0x7f22aed91150>
```



In [ ]:

```
```