Escalabilidade e Eficiência no Processamento de Grandes Catálogos Astronômicos com Dask

Luigi Silva (LIneA)
luigi.silva@linea.org.br

Objetivos

- Apresentar o uso do Dask como ferramenta escalável para lidar com grandes catálogos astronômicos em ambientes HPC, destacando seu uso prático dentro da infraestrutura do LIneA.

Motivação

- O volume de dados astronômicos vem crescendo de forma exponencial.
- Exemplos de levantamentos fotométricos:
  - DES DR2 (~5000 deg²): ~691 milhões de objetos;
  - LSST DP0.2 (~300 deg²) (Simulado): ~278 milhões de objetos.
- Para o LSST DR11 (~20000 deg²) (*full survey*), esperamos ~37 bilhões de objetos, ou seja, estamos falando de um aumento em ordem de grandeza.
- Processamento com ferramentas tradicionais (ex: Pandas) se torna inviável.
- Precisamos de soluções que escalem com os dados, aproveitando clusters e paralelismo.

DES DR2: https://arxiv.org/pdf/2101.05765
LSST DR11: https://www.lsst.org/scientists/keynumbers

LIneA

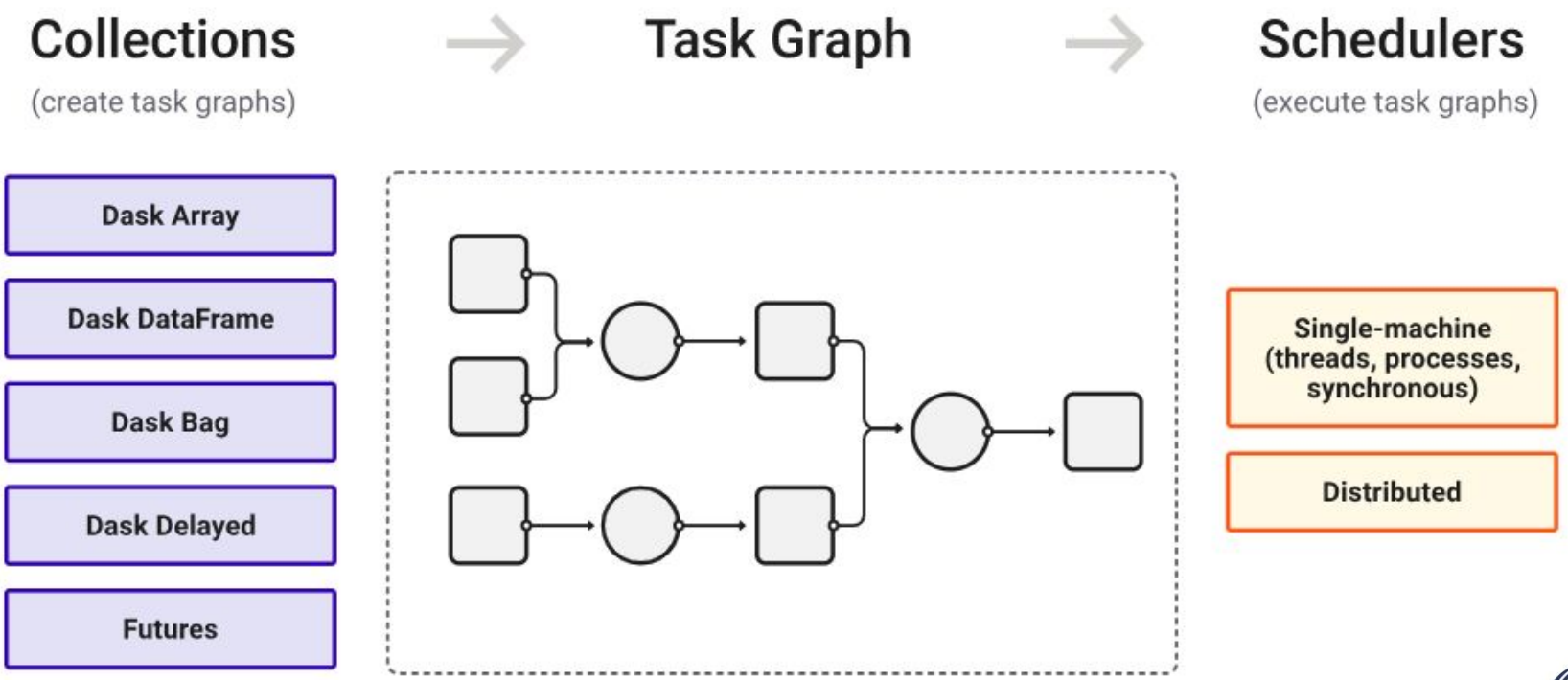
O que é o Dask?

- Biblioteca Python para computação paralela.
- Oferece estruturas como:
  - `dask.dataframe` (paralelo ao Pandas);
  - `dask.array`, `dask.bag`, `dask.delayed`, etc.
- Planejamento dinâmico de tarefas (task graph).

https://docs.dask.org/en/stable/

LIneA

O que é o Dask?



https://docs.dask.org/en/stable/10-minutes-to-dask.html

LineA

O que é o Dask?

- Executa em:
  - Máquina local com múltiplos cores;
  - Cluster HPC (ex: via SLURM);
  - Nuvem (AWS, GCP, etc.).

https://docs.dask.org/en/stable/

LIneA

Dask vs Pandas

	Pandas	Dask
Escopo de dados	Memória RAM	Dados maiores que a RAM
Execução	Sequencial	Paralela e distribuída
Interface	Familiar	Semelhante ao Pandas
Integração com HPC	Limitada	Nativa (ex: SLURM)

Dask vs Pandas

```
# Pandas
df = pd.read_parquet("catalog.parquet")
df_filtered = df[df['mag_i'] < 26]

# Dask
df = dd.read_parquet("catalog.parquet")
df_filtered = df[df['mag_i'] < 26]
```

Dask em ambientes HPC

- Integração com SLURM via SLURMCluster.
- Criação de workers como jobs SLURM.
- Configuração de núcleos, memória e tempo de execução.
- Distribuição automática de tarefas com Client.
- Escalabilidade manual (`.scale()`) ou automática (`.adapt()`).

https://docs.dask.org/en/stable/

LineA

Dask em ambientes HPC

```
cluster = SLURMCluster(
    interface="ib0",            # Lustre interface
    queue='cpu_small',          # Name of the queue
    cores=40,                   # Number of logical cores per node
    processes=2,                # Number of dask processes per node
    memory='100GB',             # Memory per node
    walltime='04:00:00',        # Maximum execution time
)

# Scaling the cluster to use X nodes
cluster.scale(jobs=10)

# Defining the dask client
client = Client(cluster)
```

Dask em ambientes HPC

- Relatórios HTML via `performance_report`.
- Métricas detalhadas: memória, CPU, gargalos e tarefas.
- Ideal para análise e otimização pós-execução.

https://docs.dask.org/en/stable/

LineA

Dask em ambientes HPC

Summary | Task Stream | System | Scheduler Logs | Worker Profile (compute) | Worker Profile (administrative) | Scheduler Profile (administrative) | Bandwidth (Workers) | Bandwidth (Types)

Dask Performance Report

Select different tabs on the top for additional information

Duration: 10m 23s

Tasks Information

- number of tasks: 6073
- compute time: 5hr 39m
- disk-read time: 12.88 ms

Scheduler Information

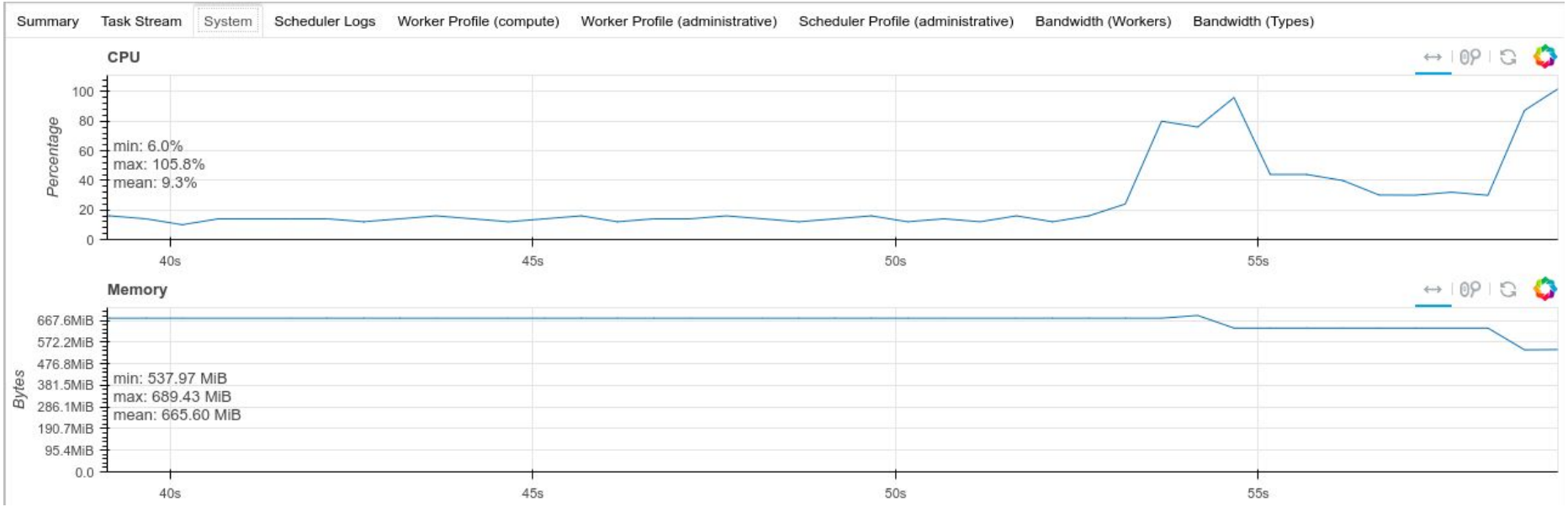
- Address: tcp://10.148.0.32:34581
- Workers: 40
- Threads: 960
- Memory: 2.07 TiB
- Dask Version: 2025.1.0
- Dask.Distributed Version: 2025.1.0

Calling Code

```
if run_the_pipeline==True:
    current_date = datetime.now().strftime('%Y-%m-%d_%H-%M')

    ################################## INPUT CONFIGS ##################################
```

Dask em ambientes HPC



Plataforma OnDemand do LIneA

- Interface web para acesso direto ao HPC do LIneA (Cluster Apollo).
- Permite criar sessões interativas de Jupyter Notebook.
- Integração com conda environments e o sistema de arquivos Lustre.
- Ideal para usar Dask em ambiente controlado, com escalabilidade.
- Vamos lá?
- https://ondemand.linea.org.br/

docs.linea.org.br/processamento/uso/openondemand.html

ondemand.linea.org.br/node/apl15/59063/lab/tree/workshop_presentation/workshop-notebook.ipynb

File Edit View Run Kernel Tabs Settings Help

Filter files by name

/ workshop_presentation /

Name	Last Modified
workshop-notebook.ipynb	16 minutes ago

Launcher | workshop-notebook.ipynb

Markdown

hats_env_290125

Workshop Notebook - Simple Test

Simple test of reading and filtering a catalog.

Author: Luigi Lucas de Carvalho Silva (luigi.lcsilva@gmail.com)

Imports

[1]:
```
########################## GENERAL ##########################
import os
import sys
import glob
import time
import numpy as np
import pandas as pd
import pyarrow.parquet as pq
########################## DASK ##########################
from dask import dataframe as dd
from dask.distributed import Client, wait
from dask_jobqueue import SLURMCluster
```

Cluster Configs

[2]:
```
interface="ib0"
queue='cpu_small'
cores=20
processes=2
memory='10GB'
walltime='04:00:00'
number_of_nodes=2

job_extra_directives=[
    '--propagate',
    f'--output=/dev/null',
    f'--error=/dev/null'
]
```

Simple 1 0 hats_env_290125 | Idle

Mode: Command Ln 1, Col 1 workshop-notebook.ipynb

Exemplo de uso real

- Leitura parcial do catálogo LSST DP0.2 (Object) Skinny. Vamos ler 109 dos 2215 arquivos .parquet.
- 7,714,406 linhas, 16 colunas, 1.09 GB.
- Vamos filtrar a magnitude da banda i para valores menores que 26 e printar o `.head()` do dataframe e ver quanto tempo isso leva usando pandas e dask.

Exemplo de uso real

```
def filter_pandas():
     df = pd.concat([pd.read_parquet(f) for f in parquet_files],
ignore_index=True)
    return df[df["mag_i"] < 26].head()

def filter_dask():
    df = dd.read_parquet(parquet_files)
    return df[df["mag_i"] < 26].head()

# Requested resources for Dask
cores=20
processes=2
memory='10GB'
number_of_nodes=2
```

Exemplo de uso real

- Com Pandas: **3.15 s**.
- Com Dask: **0.66 s**.
- O ganho de tempo é significativo, mesmo usando só dois nós.
- Mas não é só isso. Lembre-se: se o tamanho dos dados fosse maior que a RAM local disponível, o método usando o Pandas nem iria funcionar.
- Por isso não tem como fazer essa comparação para todos os dados do catálogo.

Limitações

- Nem todas as operações do Pandas são suportadas (ex: .sort_values() global, .apply() complexos).
- Overhead em tarefas muito pequenas — ideal para workloads de médio a grande porte.
- Depuração pode ser desafiadora em ambiente distribuído.
- O Dask não é mágico:
  - Pode consumir mais memória do que o esperado.
  - Isso acontece especialmente em `joins` grandes, `groupbys` ou lógicas mal particionadas.
- Se acontecer, é importante monitorar os workers e repensar:
  - O particionamento.
  - A lógica das operações.
- Uso distribuído exige conhecimento básico de HPC (SLURM, workers, ambientes).

LineA

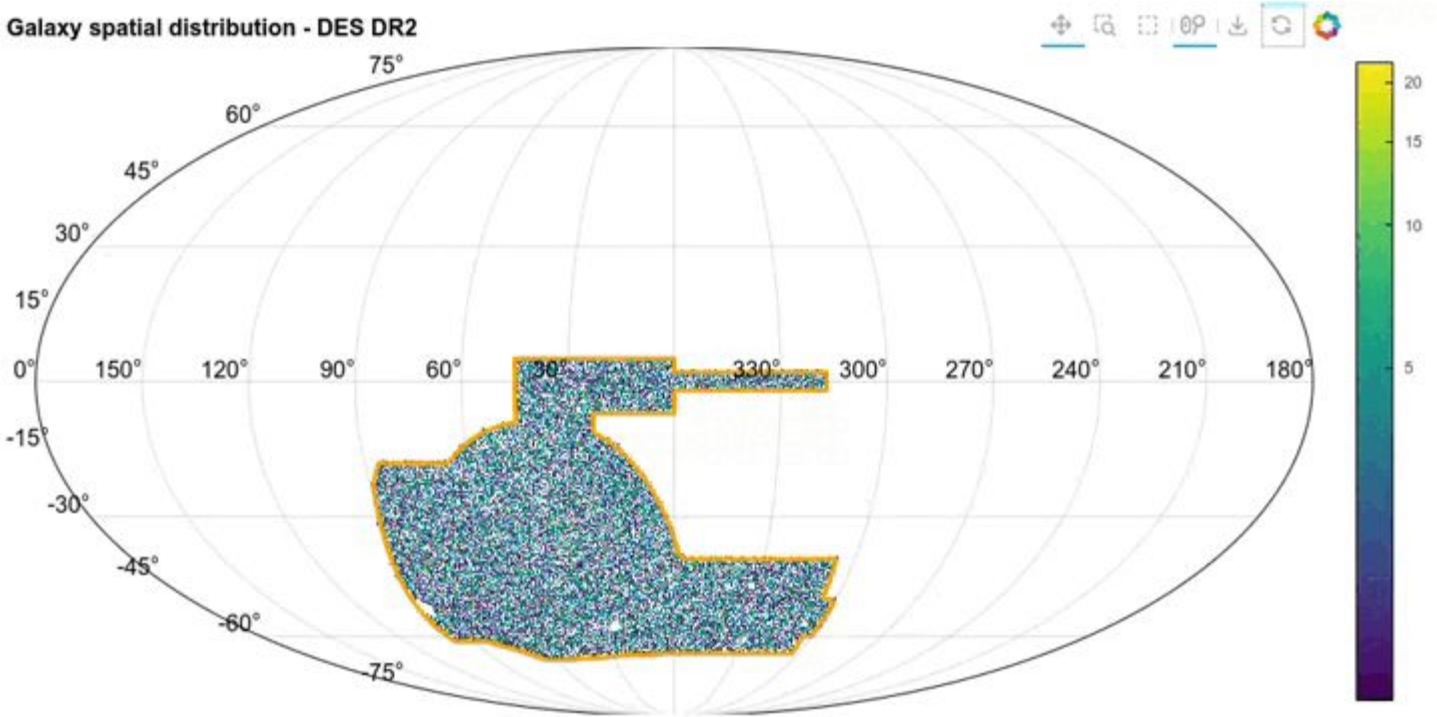
Conclusão

- O volume de dados astronômicos exige ferramentas escaláveis e eficientes.
- O Dask oferece uma solução paralela e distribuída, com API familiar para usuários de Pandas.
- Permite processar catálogos com bilhões de linhas.
- Integra-se perfeitamente com clusters HPC via SLURM, como no LIneA
- A plataforma OnDemand facilita o uso interativo do Dask no ambiente HPC.
- Ferramentas como `performance_report` ajudam a entender e otimizar o desempenho.
- Com atenção à lógica do código e ao particionamento dos dados, o Dask se mostra uma solução robusta e prática para a astronomia moderna.

Muito obrigado!
Perguntas?

Extras!

Visualização de dados com Dask e Datashader - TBD



Notebook com exemplo de plot com Dask e Datashader integrados:
https://github.com/linea-it/pz-compute/blob/main/doc/dp02_duplicates/3_DP02_duplicates_spatial_distribution.ipynb