



# Dasking in the Sun

---

**Applying Parallel Operations to your Pandas Transformations**

What do you do when your  
data become too large? Is it  
CPU-bound? Or memory-  
bound?

---

# Dask docs provide guidance

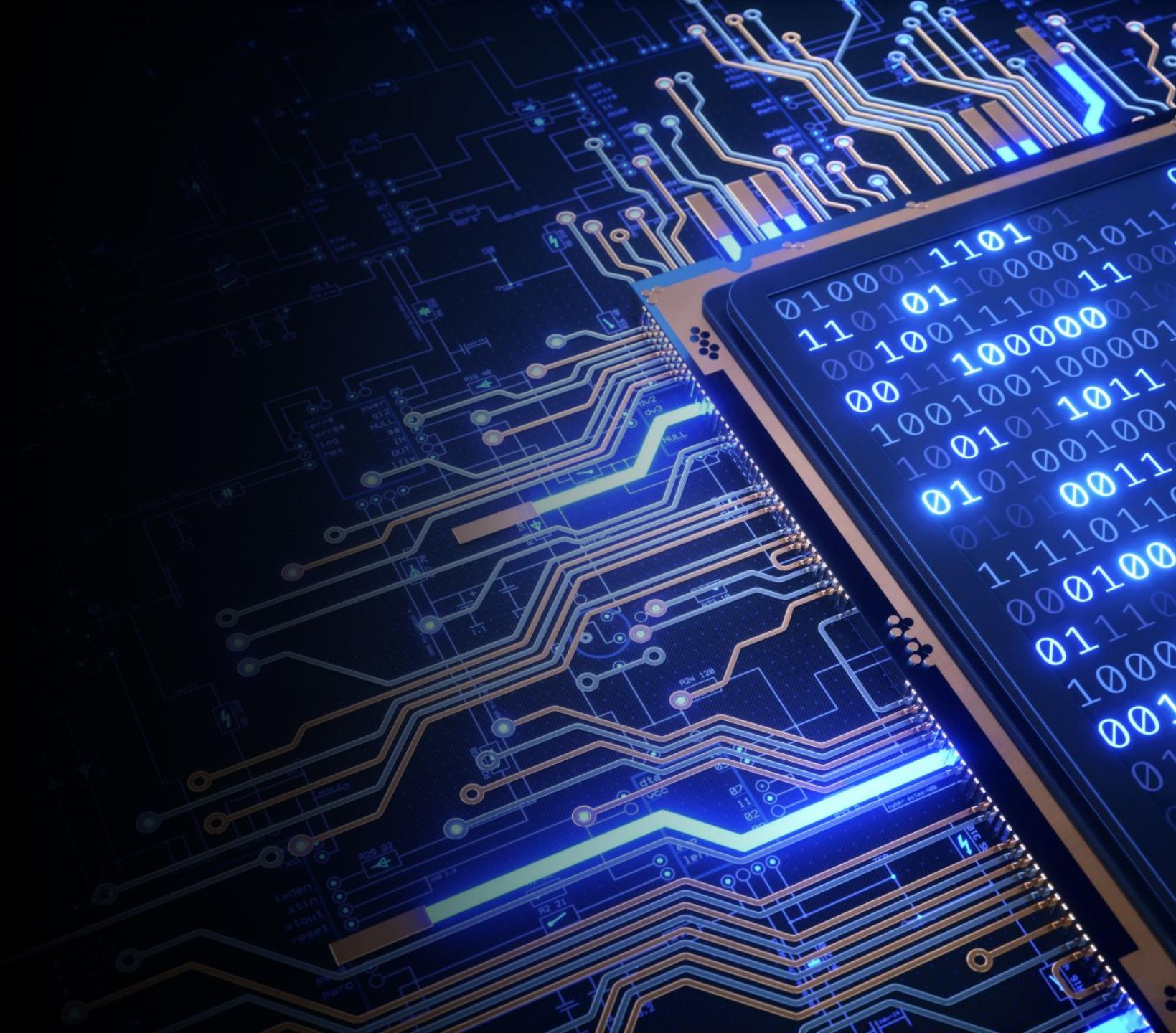
The Dask documentation is organized into several main sections:

- Dask Tutorial**: A search bar at the top.
- Parallel and Distributed Machine Learning**: This section is currently selected, indicated by a red border around its title.
- Types of Scaling**: A sub-section under Parallel and Distributed Machine Learning.
- Training on Large Datasets**: A section at the bottom of the sidebar.
- Get Started**, **Algorithms**, **Setup**, and **Community**: Navigation links at the top right.

Content on the right side of the page includes:

- A note about running the notebook in a live session or on GitHub.
- The title **Parallel and Distributed Machine Learning**.
- A link to **Dask-ML** resources.
- A section titled **Types of Scaling** with a sub-section icon.
- A note about scaling problems: "There are a couple of distinct scaling problems you might face. The scaling strategy depends on which problem you're facing."
- A numbered list of scaling types:
  1. CPU-Bound: Data fits in RAM, but training takes too long. Many hyperparameter combinations, a large ensemble of many models, etc.
  2. Memory-bound: Data is larger than RAM, and sampling isn't an option.
- A diagram illustrating hyperparameter optimization. It shows a central node labeled "Choose Best Parameters" connected to six "SGDClassifier" nodes, each with a specific alpha value. Below each SGDClassifier node is a "TfidfTransformer" node with a specific norm value, indicating a pipeline structure.

- 
- **RAM:** Your computer's short-term data storage. Can be accessed quickly.
  - **Cores:** Modern CPUs offer cores and hyper-threading. A dual-core has 2 CPUs, quad-core has 4 CPUs, octo-core has 8, and so on.
  - Processes tend to be memory-bound or CPU-bound, meaning that either the data are very large and you might run out of memory, or the number of computations is so great that you might run out of CPU.



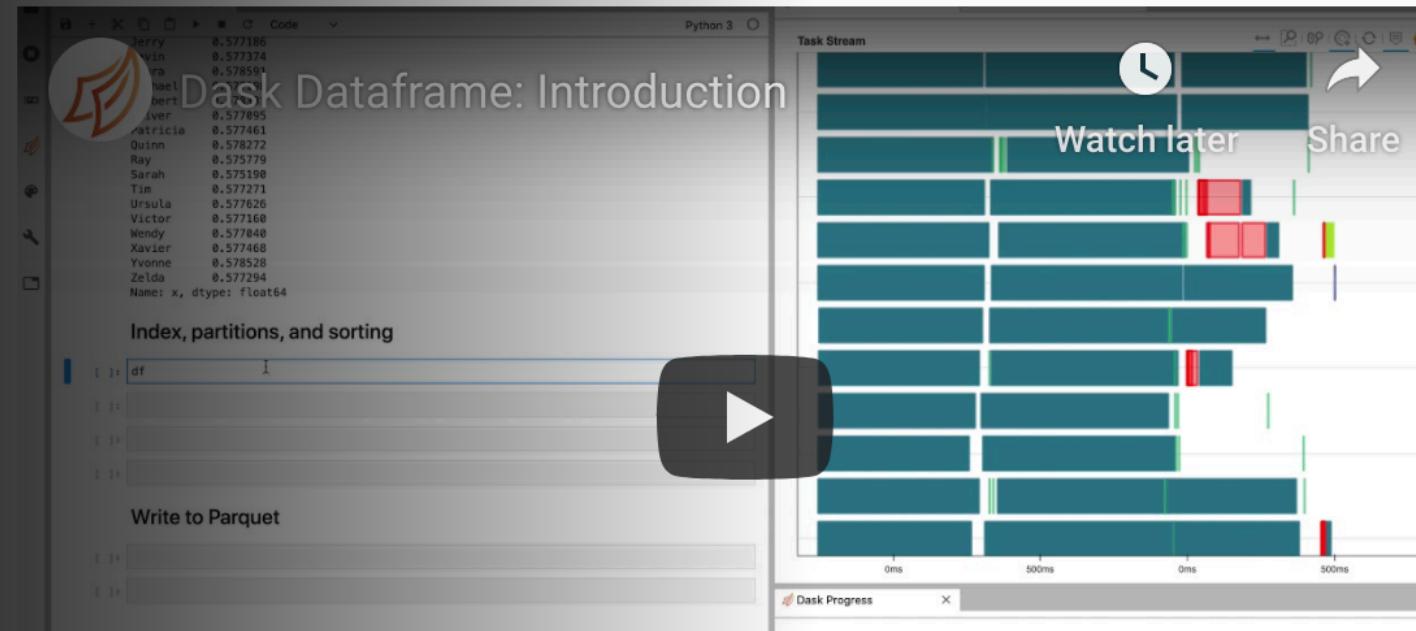


- Today's code was run on:
- Python virtualenv created with venv
- Requirements are in requirements.txt
- Python 3.7.6
- MacOS

# DASK

## DataFrame

A Dask DataFrame is a large parallel DataFrame composed of many smaller Pandas DataFrames split along the index. These Pandas DataFrames may live on disk for larger-than-memory data, on a single machine, or on many different machines in a cluster. One Dask DataFrame operation triggers many operations on the constituent Pandas DataFrames.





- The basic concept of Dask is to split data/computation up across a computer's CPU.

- Dask is installable:

```
$conda install dask
```

```
$pip install dask
```

- Or from source:

```
$git clone https://github.com/dask/dask.git
```

```
$cd dask
```

```
$python setup.py install
```

## Dask

- Can scale from a laptop to a cluster
- Has the concept of arrays and dataframes
- Has several ways of distributing tasks and scheduling them, including dask.delayed
- Has dask-ml, parallelized machine learning that works alongside scikit-learn (scikit-learn already has joblib, but Dask extends this package to clusters)

[Get Started](#)[Algorithms](#)[Setup](#)[Community](#)

Dataframe
Series
Groupby Operations
Rolling Operations
Create DataFrames
Store DataFrames
Convert DataFrames
Reshape DataFrames
DataFrame Methods
Series Methods
DataFrameGroupBy
SeriesGroupBy
Custom Aggregation
Storage and Conversion
Rolling
Resampling
Dask Metadata
Other functions
Create and Store Dask DataFrames
Best Practices
Design

# API

## Dataframe

<code>DataFrame(dsk, name, meta, divisions)</code>	Parallel Pandas DataFrame
<code>DataFrame.add(other[, axis, level, fill_value])</code>	Get Addition of dataframe and other, element-wise (binary operator <code>add</code> ).
<code>DataFrame.append(other[, interleave_partitions])</code>	Append rows of <code>other</code> to the end of caller, returning a new object.
<code>DataFrame.apply(func[, axis, broadcast, ...])</code>	Parallel version of <code>pandas.DataFrame.apply</code>
<code>DataFrame.assign(**kwargs)</code>	Assign new columns to a DataFrame.
<code>DataFrame.astype(dtype)</code>	Cast a pandas object to a specified dtype <code>dtype</code> .
<code>DataFrame.categorize([columns, index, ...])</code>	Convert columns of the DataFrame to category dtype.
<code>DataFrame.columns</code>	
<code>DataFrame.compute(**kwargs)</code>	Compute this dask collection
<code>DataFrame.corr([method, min_periods, ...])</code>	Compute pairwise correlation of columns, excluding NA/null values.
<code>DataFrame.count([axis, split_every])</code>	Count non-NA cells for each column or row

- 
- Multiprocess: <https://docs.python.org/3/library/multiprocessing.html>
  - Modin: <https://modin.readthedocs.io/en/latest/>
  - Swifter: <https://pypi.org/project/swifter/>
  - Ray: <https://pypi.org/project/ray/>
  - Pandarallel: <https://pypi.org/project/pandarallel/>
  - Dask: <https://dask.org/>

- 
- vectorization
  - .iterrows()
  - .apply()
  - .itertuples()
  - <https://medium.com/swlh/why-pandas-itertuples-is-faster-than-iterrows-and-how-to-make-it-even-faster-bc50c0edd30d>



Methodology	Average single run time	Marginal performance improvement
Crude looping	645 ms	
Looping with iterrows()	166 ms	3.9x
Looping with apply()	90.6 ms	1.8x
Vectorization with Pandas series	1.62 ms	55.9x
Vectorization with NumPy arrays	0.37 ms	4.4x



Top highlight

<https://engineering.upside.com/a-beginners-guide-to-optimizing-pandas-code-for-speed-c09ef2c6a4d6>