Week 08: Dask
DS-GA 1004: Big Data
Detailed Notes for Final Exams

Introduction to Dask

Dask is an open-source Python library for parallel computing. Designed to scale from single machines to clusters, it integrates with the Python scientific stack (NumPy, Pandas, Scikit-Learn) and supports out-of-core computation.

Key Features

- **Delayed Computation**: Builds task graphs for lazy evaluation (similar to Spark RDDs).
- **Collections**: Provides distributed versions of common data structures:
  - **Bags**: Unstructured data (parallel Python lists).
  - **DataFrames**: Tabular data (Pandas-like, partitioned).
  - **Arrays**: N-dimensional arrays (NumPy-like, chunked).
- **Out-of-Core Processing**: Handles datasets larger than RAM by chunking data.

Comparison of Big Data Frameworks

Method	**Strengths**	**Weaknesses**
Collections of Files	Flexible, unstructured	No built-in parallelism
Relational Databases	SQL interface, structured	Complex parallelism
Map-Reduce + HDFS	Parallel, scalable	Restricted to map/reduce
Spark	Mature, SQL/DataFrames, cluster-ready	Poor Python integration, rigid data model
Dask	Python-native, out-of-core, flexible	Less mature, requires manual optimization

Dask vs. Spark

Similarities

- Lazy evaluation via task graphs.
- Distributed DataFrames and collections.
- Fault tolerance through lineage.

Differences

Aspect	Dask	Spark
Language	Python-centric	JVM-based (Scala/Java)
Data Model	Prioritizes arrays (NumPy)	Prioritizes tabular (SQL)
Scaling	Single-machine out-of-core + clusters	Cluster-first
Ecosystem	SciPy stack (sklearn, PyTorch)	Hadoop ecosystem (HDFS, YARN)

Dask Collections

Bags

- Unordered collections of Python objects (analogous to Spark RDDs).
- Operations: `map`, `filter`, `foldby`.
- Example:

```
import dask.bag as db
b = db.from_sequence(range(5))
c = b.map(lambda x: x**2)
c.compute()  # [0, 1, 4, 9, 16]
```

- **Optimization Tip**: Avoid `groupby` (high shuffle); use `foldby` for associative/commutative operations.

DataFrames

- Partitioned Pandas DataFrames. Read from CSV/Parquet:

```
import dask.dataframe as dd
df = dd.read_csv('s3://bucket/*.csv')
df.groupby('column').mean().compute()
```

- **Partition Management**:
  - Repartition after filtering to avoid empty partitions.
  - Use `repartition()` and `persist()` for balanced workloads.

Arrays

- Chunked NumPy arrays for large datasets:

```
import dask.array as da
x = da.from_array(np.random.randn(2000, 6000), chunks
    =(1000, 1000))
```

- Supports most NumPy operations (e.g., slicing, reductions).

Schedulers and Execution

- **Single Machine**:
  - Threads: Shared memory, lightweight.
  - Processes: Avoid GIL, true parallelism.
- **Clusters**: Deploy on YARN, Kubernetes, or HPC systems (e.g., Greene).

Case Study: Machine Listening Evaluation

- **Problem**: Evaluate 20,000 model outputs across 10 models and 2,000 audio files.
- **Solution with Dask**:
  1. Store outputs as {`model_id`}/{`recording_id`}`.txt`.
  2. Use delayed functions for parallel scoring:

```
from dask import delayed
@delayed
def evaluate(file):
    # Load data, compute metrics
    return metrics
results = [evaluate(f) for f in glob('*/*.txt')]
df = dd.from_delayed(results).compute()
```

  3. Convert to DataFrame and save as Parquet.
- **Why Dask?**: Embarrassingly parallel, minimal code changes.

Best Practices

- **Minimize Shuffling**: Use combiners (`foldby`) and avoid wide dependencies.
- **Chunk Sizing**: Balance chunk size (too small → overhead; too large → memory issues).

- **Cluster vs. Single Machine**: Use clusters only when data exceeds single-node resources.

Alternatives to Dask

- **Polars**: Rust-based parallel DataFrame library (columnar, multi-threaded).
- **Modin**: Distributed Pandas replacement.

Exam Tips

- Understand when to use Dask vs. Spark (Python integration vs. mature ecosystem).
- Know how Dask handles out-of-core computation (chunking + task graphs).
- Be able to contrast Bags, DataFrames, and Arrays.