Lecture 4: Dask

AC295

Advanced Practical Data Science

Pavlos Protopapas

Outline

1: Communications

2: Motivation

3: Dask API

4: Directed Acyclical Graph (DAGs)

5: Computational Resources

6: Exercise: Exploratory Data Analysis with DASK

7: Exercise: Visualize Directed Acyclic Graphs (DAGs)

8: Task Scheduling

9: Exercise: Manipulate Structured Data

10: Limitations

Outline

1: Communications

Communications

Feedback from week 2 reading

On a scale of 1 (very poor) to 5 (excellent), how would you rate the quality of this week's readings?

A	1	0%
B	2	11%
C	3	22%
D	4	56%
E	5	11%

Communications

Feedback from week 2 reading

3
6 hours
1
2 hours
1.5
2
4-5
2
around 5-6 hours? (I have to read and google at the same time so it might take a little bit longer

Communications

Feedback from week 2 reading

n/a
It would be better to have more office hours for people with little experience with course materials to ask for help.
No
great!
I'm enjoying it! And hoping the practicum isn't too hard. It's unclear to me how much experience others have in these topics; I'm simultaneously concerned that I have a lot less understanding than the rest of the class and pretty sure others in the class are just as lost as me. I guess it would be nice to see how others in the class actually feel about this stuff.
The course has more parts and pieces than any other class I've taken. Exercises, practicums, lectures, projects, quizzes, etc. It's a lot to keep track of. Otherwise it's good.
like the discussion style of the class. more info one what tech companies are using and how would be great.
0.2
I love the materials and hope that I can get familiar with the terms as soon as possbile

Outline

1: Communications

2: Motivations

Motivation

Competitive Advantage

Computing More Powerful

How to enable it?

From Analytics to Data Science Driven

Increasing Data

Dask API

Dataset type	Size range	Fits in RAM?	Fits on local disk?
Small dataset	Less than 2-4 GB	Yes	Yes
Medium dataset	Less than 2 TB	No	Yes
Large dataset	Greater than 2 TB	No	No

Adapted from Data Science with Dask

Outline

1: Communications

2: Motivations

3: Dask API

Dask API

What is **unique** about Dask:

- allow to **work with larger datasets** making it possible to parallelize computation (e.g. "simple" sorting and aggregating functions would otherwise spill on persistent memory).
- it **simplify** the cost of using more complex **infrastructure**.
- it is **easy to learn** for data scientists with a background in the Python (similar syntax) and flexible.

Adapted from Data Science with Dask

Dask API

- Dask is fully implemented in Python and natively scales NumPy, Pandas, and scikit-learn.
- Dask can be used effectively to work with both medium datasets on a single machine and large datasets on a cluster.
- Dask can be used as a general framework for parallelizing most Python objects.
- Dask has a very low configuration and maintenance overhead.

Adapted from Data Science with Dask

Dask API <cont>

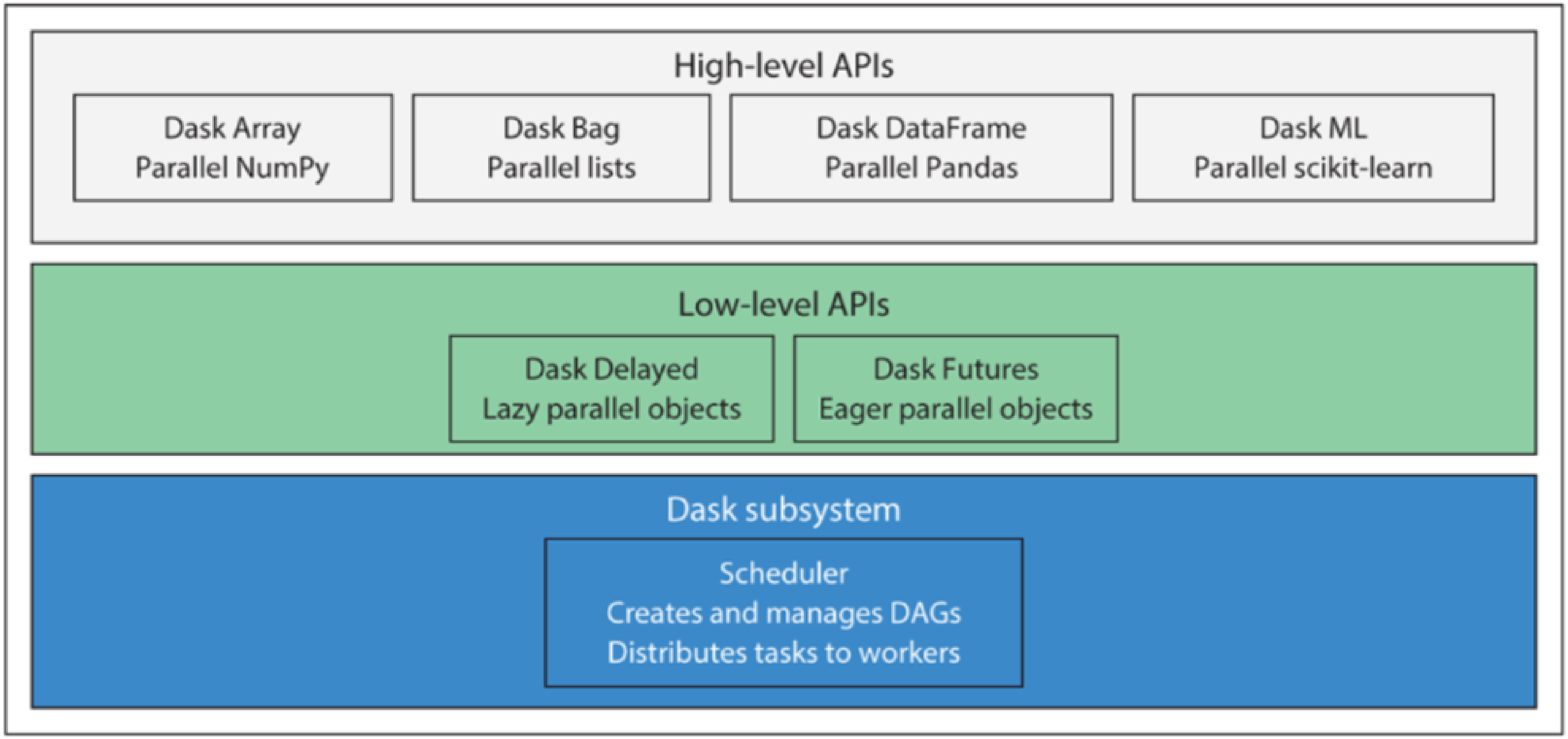
It helps applying distributed computing to data science project:

- **not of great help for small size datasets**: It would generate greater overheads. Complex operations can be done without spilling to disk and slowing down process.

- **very useful for medium size dataset:** it allows to work with medium size in local machine. Difficult to take advantage of parallelism within Pandas (no sharing work between processes on multicore systems).

- **essential for large datasets**: Pandas, NumPy, and scikit-learn are not suitable at all for datasets of this size, because they were not inherently built to operate on distributed datasets.

Dask API <cont>



Adapted from Data Science with Dask

Outline

1: Communications

2: Motivation

3: Dask API

4: Directed Acyclical Graph (DAGs)

Directed Acyclical Graph

A graph is a representation of a **set of objects that have a relationship** with one another. It is good to represent a wide variety of information.

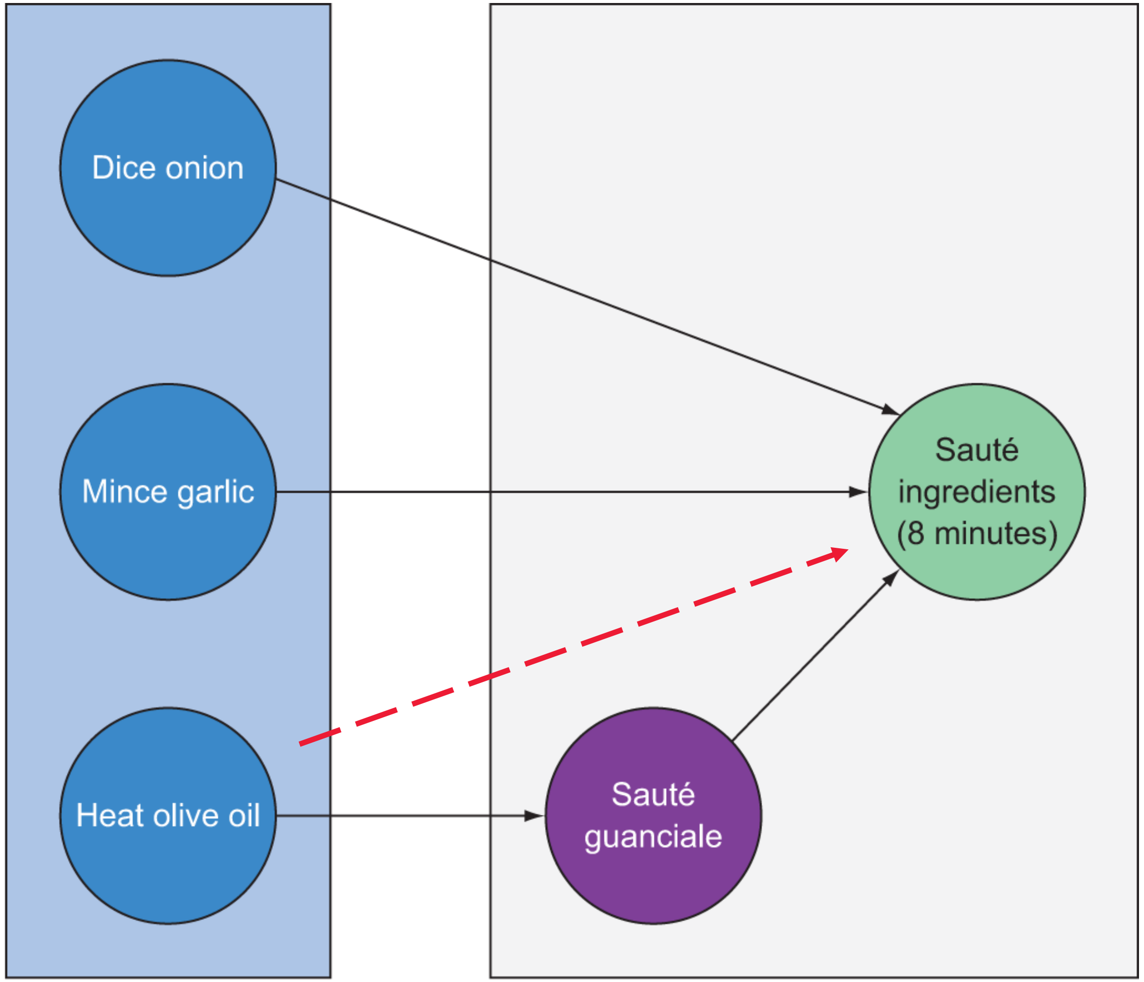
A graph is compounded by:

- **node**: a function, an object or an action
- **line**: symbolize the relationship among nodes

In a directed acyclical graph there **is one logical way to traverse** the graph. No node is visited twice.

In a cyclical graph: exist a feedback loop that allow to revisit and repeat the actions within the same node.

Directed Acyclical Graph <cont>

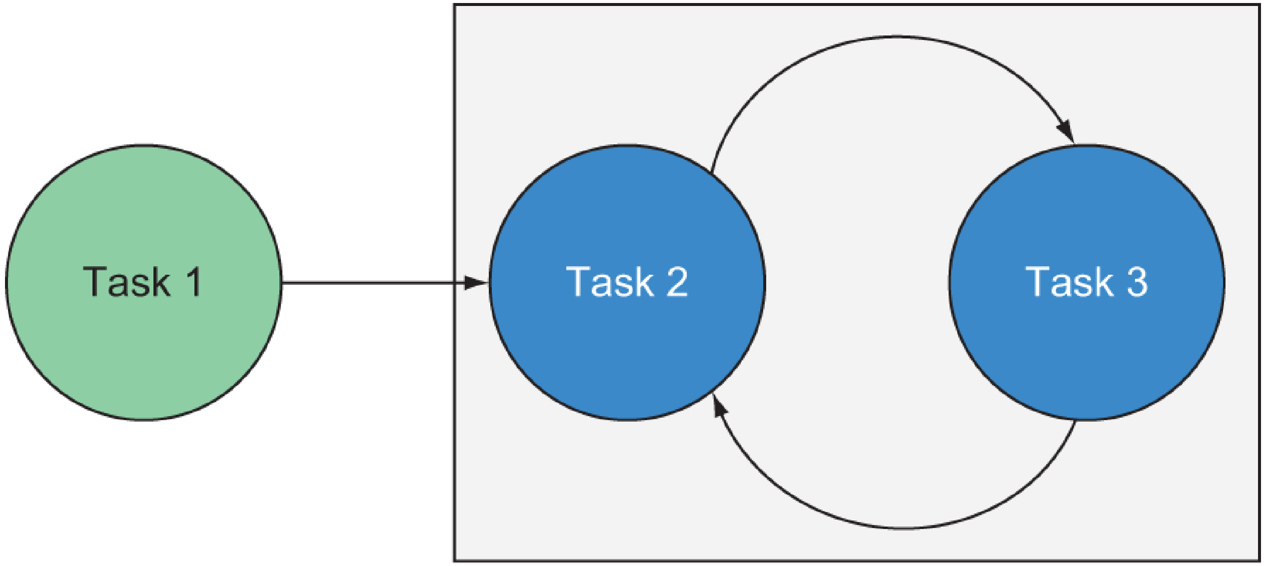


No dependencies. These tasks can be started in any order.

These tasks can only be started when all nodes connected to them have been completed.

Adapted from
Data Science with Dask

Directed ~~A~~cyclical Graph <cont>



Task 2 and Task 3 are connected to each other in an infinite feedback loop. There is no logical termination point in this graph.

Adapted from
Data Science with Dask

Outline

1: Communications

2: Motivation

3: Dask API

4: Directed Acyclical Graph (DAGs)

5: Computational Resources

Computational Resources

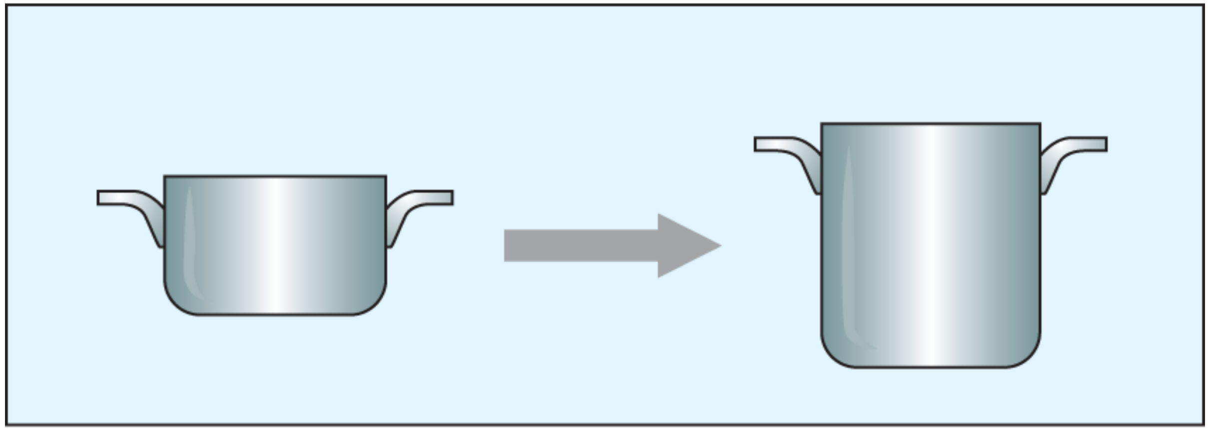
How to handle computational resources? As the problem we solve requires more resources we have two options:

- **scale up**: increase size of the available resource: invest in more efficient technology. **cons** diminishing return.

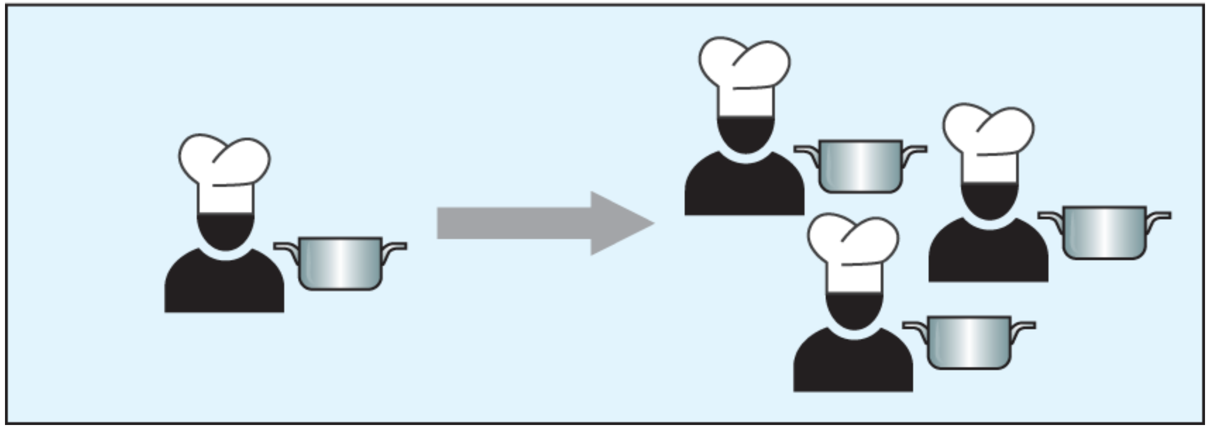
- **scale out**: add other resources (dask's main idea). Invest in more cheap resources. **cons** distribute workload.

Computational Resources <cont>

Scale up



Scale out



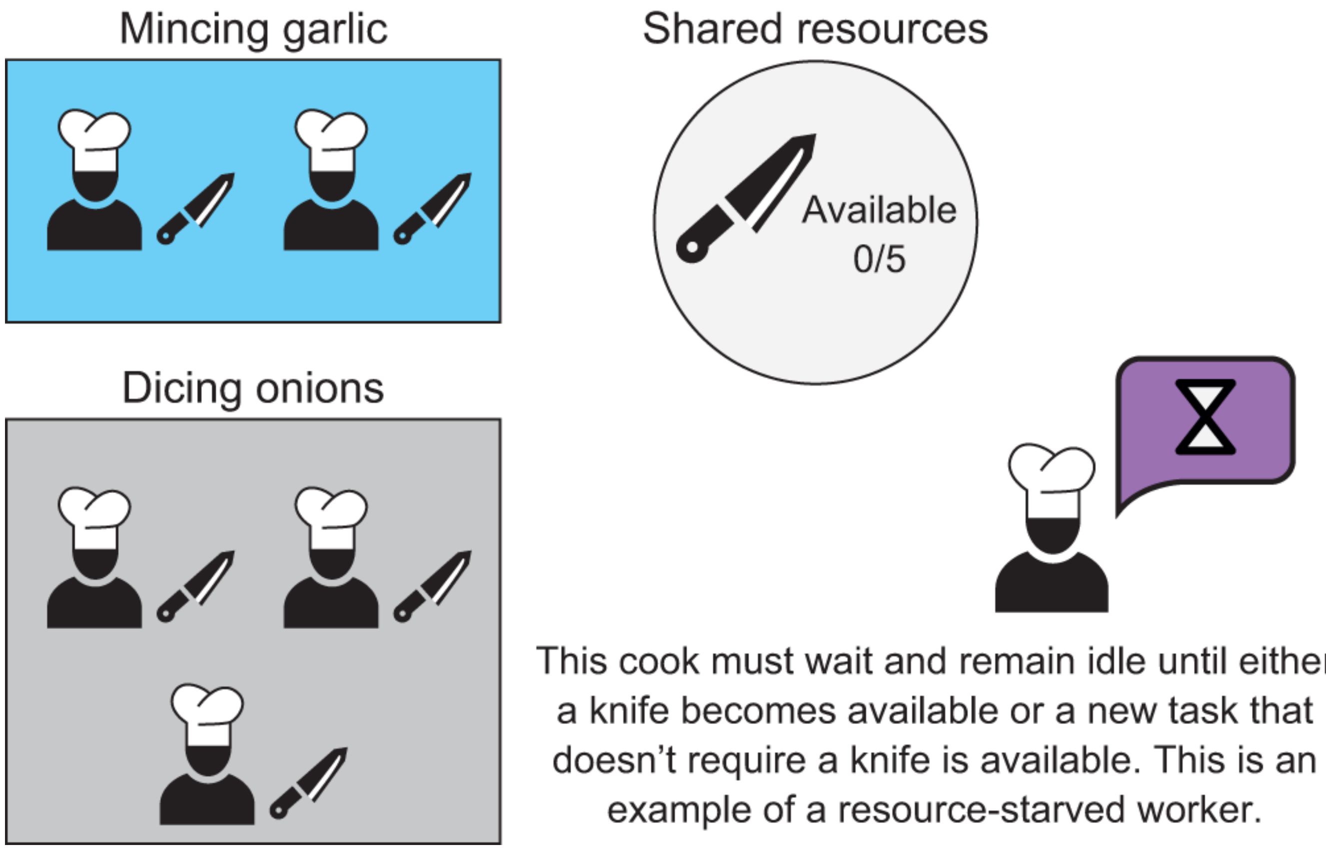
Adapted from
Data Science with Dask

Computational Resources <cont>

As we approach greater number of "work to be completed", some resources might be not fully exploited. This phenomenon is called **concurrency.**

For instance some might be idling because of insufficient shared resources (i.e. *resource starvation*). Schedulers handle this issue by making sure to provide enough resources to each worker.

Computational Resources <cont>



This cook must wait and remain idle until either a knife becomes available or a new task that doesn't require a knife is available. This is an example of a resource-starved worker.

Adapted from
Data Science with Dask

Computational Resources <cont>

In case of a failure, Dask reach a node and repeat the action without disturbing the rest of the process. There are two types of failures:

- **work failures**: a worker leave, and you know that you must assign another one to their task. This might potentially slow down the execution, however it won't affect previous work (aka data loss).

- **data loss**: some accident happens, and you have to start from the beginning. The scheduler stops and restarts from the beginning the whole process.

Dask Review

- Dask can be used to scale popular Python libraries such as Pandas and NumPy allowing **to analyze dataset with greater size** (>8GB).
- Dask uses **directed acyclical graph to coordinate execution** of parallelized code across processors.
- Upstream actions are completed before downstream nodes.
- **Scaling out** (i.e. add workers) can improve performances of complex workloads, however, create overhead that can reduces gains.
- In case of failure, the step to reach a **node can be repeated** from the beginning without disturbing the rest of the process.

Outline

1: Communications

2: Motivation

3: Dask API

4: Directed Acyclical Graph (DAGs)

5: Computational Resources

6: Exercise: Exploratory Data Analysis with DASK

Outline

1: Communications

2: Motivation

3: Dask API

4: Directed Acyclical Graph (DAGs)

5: Computational Resources

6: Exercise: Exploratory Data Analysis with DASK

7: Exercise: Visualize Directed Acyclic Graphs (DAGs)

Outline

1: Communications

2: Motivation

3: Dask API

4: Directed Acyclical Graph (DAGs)

5: Computational Resources

6: Exercise: Exploratory Data Analysis with DASK

7: Exercise: Visualize Directed Acyclic Graphs (DAGs)

8: Task Scheduling

Task scheduling

- Dask performs the so called lazy computations. Until you run the method .compute() all what Dask does is to split the process into smaller logical pieces (save memory).

- Even though the process is defined, the number of resources assigned and the place where the result will be stored are note assigned because the scheduler assign them dynamically. This allow to recover from worker failure.

Task scheduling <cont>

- Dask uses a central scheduler to orchestrate the work. It splits the workload among different servers which unlikely they are perfectly the same for load, power or access to data. Due to these conditions, scheduler needs to promptly react to avoid bottlenecks that will affect overall runtime.

- For best performance, a Dask cluster should use a distributed file system (S3, HDFS) to back its data storage. Assuming there are two nodes like in the image below and data are stored in one. In order to perform computation in the other node we have to move the data from one to the other creating an overhead proportional to the size of the data. The remedy is to split data minimizing the number of data to broadcast across different local machines.

Outline

1: Communications

2: Motivation

3: Dask API

4: Directed Acyclical Graph (DAGs)

5: Computational Resources

6: Exercise: Exploratory Data Analysis with DASK

7: Exercise: Visualize Directed Acyclic Graphs (DAGs)

8: Task Scheduling

9: Exercise: Manipulate Structured Data

Outline

1: Communications

2: Motivations

3: Dask API

4: Directed Acyclical Graph (DAGs)

5: Computational Resources

6: Exercise: Exploratory Data Analysis with DASK

7: Exercise: Visualize Directed Acyclic Graphs (DAGs)

8: Task Scheduling

9: Exercise: Manipulate Structured Data

10: Dask Review and Limitations

Dask Review

- Dask can be used to scale popular Python libraries such as Pandas and NumPy allowing **to analyze dataset with greater size** (>8GB).
- Dask uses **directed acyclical graph to coordinate execution** of parallelized code across processors.
- Upstream actions are completed before downstream nodes.
- **Scaling out** (i.e. add workers) can improve performances of complex workloads, however, create overhead that can reduces gains.
- In case of failure, the step to reach a **node can be repeated** from the beginning without disturbing the rest of the process.

Dask Limitations

- Dask dataframe are immutable. Functions such as pop and insert are not supported.
- Does not allow for functions with a lot of shuffling like stack/unstack and melt.
- Limit this operations after major filter and preprocessing.
- Join, merge, groupby, and rolling are supported but expensive due to shuffling
- Reset index starts sequential counting for each partitions.
- Apply and iterrow are known to be inefficient in Pandas, the same for Dask
- Use .division() to inspect how DataFrame has been partitioned.
- For best performance partitions should be roughly equal. Use .repartition() method to balance across datasets.
- For best performance sort by logical columns, partition by index and index should be presorted.

THANK YOU