# User

See my below xlwings code:

@xw.func @xw.arg('days\_amounts', ndim=2) def aging\_buckets\_parallel(days\_amounts): client = Client(n\_workers=4, threads\_per\_worker=2) df = pd.DataFrame(days\_amounts, columns=['days', 'amount']) df['days'] = df['days'].astype(int) df['amount'] = df['amount'].astype(int) df['days\_amounts'] = list(zip(df['days'], df['amount'])) ddf = dd.from\_pandas(df, npartitions=4) aging\_list = [ (lambda day, amount: amount < 0, "Advance"), (lambda day, amount: 0 <= day < 31, "Not Due"), (lambda day, amount: 31 <= day < 61, "31-60 Days"), (lambda day, amount: 61 <= day < 91, "61-90 Days"), (lambda day, amount: 91 <= day < 181, "91-180 Days"), (lambda day, amount: 181 <= day < 366, "6 Months to 1 year"), (lambda day, amount: 366 <= day, "More than 1 year") ] ddf['age\_bucket'] = ddf['days\_amounts'].map(lambda x: next((label for condition, label in aging\_list if condition(x[0], x[1])), None), meta=('days', 'object')) result\_df = ddf.compute() result = [[item] for item in result\_df['age\_bucket'].values] client.close() return result

Results into following error:

Imported functions from the following modules: xlwings\_trial 2024-02-21 19:04:55,112 - distributed.worker - WARNING - Compute Failed Key: ('map-be026d46aa1dc78d5490f20fbaf10cf9', 2) Function: map args: (506 (694, 213072) 507 (400, 326997) 508 (318, 494295) 509 (585, 34393) 510 (313, 227760) ... 754 (530, 101238) 755 (464, 245233) 756 (641, -75998) 757 (40, -6458) 758 (369, 138393) Name: days\_amounts, Length: 253, dtype: string, <function aging\_buckets\_parallel.<locals>.<lambda> at 0x00000231479993A0>, None) kwargs: {} Exception: 'TypeError("'<' not supported between instances of 'str' and 'int'")'

2024-02-21 19:04:55,119 - distributed.worker - WARNING - Compute Failed Key: ('map-be026d46aa1dc78d5490f20fbaf10cf9', 3) Function: map args: (759 (229, 136402) 760 (433, 100610) 761 (210, 67421) 762 (193, -61961) 763 (335, 248791) ... 1006 (23, -77416) 1007 (530, 60412) 1008 (623, 156195) 1009 (70, 105241) 1010 (15, 33100) Name: days\_amounts, Length: 252, dtype: string, <function aging\_buckets\_parallel.<locals>.<lambda> at 0x000001E391E45B20>, None) kwargs: {} Exception: 'TypeError("'<' not supported between instances of 'str' and 'int'")'

2024-02-21 19:04:55,126 - distributed.worker - WARNING - Compute Failed Key: ('map-be026d46aa1dc78d5490f20fbaf10cf9', 0) Function: map args: (0 (236, -68848) 1 (161, -4128) 2 (165, 94371) 3 (407, 476822) 4 (486, 173170) ... 248 (591, 458860) 249 (626, 380069) 250 (603, -73775) 251 (525, 168386) 252 (426, 163054) Name: days\_amounts, Length: 253, dtype: string, <function aging\_buckets\_parallel.<locals>.<lambda> at 0x000002655700A980>, None) kwargs: {} Exception: 'TypeError("'<' not supported between instances of 'str' and 'int'")'

pythoncom error: Python error invoking COM method.

Traceback (most recent call last): Exception: 'TypeError("'<' not supported between instances of 'str' and 'int'")'

2024-02-21 19:04:55,119 - distributed.worker - WARNING - Compute Failed Key: ('map-be026d46aa1dc78d5490f20fbaf10cf9', 3) Function: map args: (759 (229, 136402) 760 (433, 100610) 761 (210, 67421) 762 (193, -61961) 763 (335, 248791) ... 1006 (23, -77416) 1007 (530, 60412) 1008 (623, 156195) 1009 (70, 105241) 1010 (15, 33100) Name: days\_amounts, Length: 252, dtype: string, <function aging\_buckets\_parallel.<locals>.<lambda> at 0x000001E391E45B20>, None) kwargs: {} Exception: 'TypeError("'<' not supported between instances of 'str' and 'int'")'

2024-02-21 19:04:55,126 - distributed.worker - WARNING - Compute Failed Key: ('map-be026d46aa1dc78d549

# Bing

The error message you're seeing is due to a TypeError that's being raised because the '<' operator is not supported between instances of 'str' and 'int'. This error is occurring in your map function where you're trying to compare a string with an integer.

The issue seems to be with the data type of the 'days' and 'amount' fields in your DataFrame. Although you've attempted to convert these fields to integers using the astype(int) method, the error message suggests that they are still being treated as strings during the comparison operation in your map function.

One possible reason for this could be that the 'days' and 'amount' fields in your DataFrame contain non-numeric values or NaNs, which cannot be converted to integers using the astype(int) method. When astype(int) fails to convert a value, it leaves the original value (which is a string) in place.

To fix this issue, you need to ensure that the 'days' and 'amount' fields in your DataFrame only contain numeric values before attempting to convert them to integers. You can do this by adding some data cleaning steps before the astype(int) calls in your code.

Here's an example of how you can clean your data:

df['days'] = pd.to\_numeric(df['days'], errors='coerce')

df['amount'] = pd.to\_numeric(df['amount'], errors='coerce')

The to\_numeric function attempts to convert non-numeric data to numbers. By setting errors='coerce', you're instructing it to replace any values that it can't convert to NaN.

After this, you can drop any rows that contain NaN values, or replace them with a value of your choice, before converting the 'days' and 'amount' fields to integers.

I hope this helps! Let me know if you have any other questions. 😊

# User

My data is like this Sample:

236 -68848 161 -4128 165 94371 407 476822 486 173170 442 162584 642 105597 361 374034 91 47504 13 75730 214 130488 375 149013 666 172780

Does negative amounts create errors?

# Bing

No, negative amounts should not create errors in your code. The issue you're facing is not related to the negative amounts but rather the comparison operation in your map function.

The error message you're seeing is due to a TypeError that's being raised because the '<' operator is not supported between instances of 'str' and 'int'. This error is occurring in your map function where you're trying to compare a string with an integer.

The issue seems to be with the data type of the 'days' and 'amount' fields in your DataFrame. Although you've attempted to convert these fields to integers using the astype(int) method, the error message suggests that they are still being treated as strings during the comparison operation in your map function.

One possible reason for this could be that the 'days' and 'amount' fields in your DataFrame contain non-numeric values or NaNs, which cannot be converted to integers using the astype(int) method. When astype(int) fails to convert a value, it leaves the original value (which is a string) in place.

To fix this issue, you need to ensure that the 'days' and 'amount' fields in your DataFrame only contain numeric values before attempting to convert them to integers. You can do this by adding some data cleaning steps before the astype(int) calls in your code.

Here's an example of how you can clean your data:

df['days'] = pd.to\_numeric(df['days'], errors='coerce')

df['amount'] = pd.to\_numeric(df['amount'], errors='coerce')

The to\_numeric function attempts to convert non-numeric data to numbers. By setting errors='coerce', you're instructing it to replace any values that it can't convert to NaN.

After this, you can drop any rows that contain NaN values, or replace them with a value of your choice, before converting the 'days' and 'amount' fields to integers.

I hope this helps! Let me know if you have any other questions. 😊

# User

One value is as big as:

488364891923092000

Can this be a problem?

# Bing

The large value you've mentioned should not inherently cause a problem. Python can handle very large integers without any issues. However, depending on the operations you're performing or the libraries you're using, there could be limitations.

For example, certain libraries or functions might have limitations on the size of integers they can handle. If you're using such a library or function, this could potentially cause a problem.

In your specific case, you're using pandas and Dask, both of which should be able to handle large integers. However, if you're performing operations that require a lot of memory (like certain types of sorting or grouping), you could potentially run into memory issues if your dataset is very large.

If you're seeing errors or unexpected behavior, it could be due to something else in your code. If you provide more details about the problem you're experiencing, I might be able to help you troubleshoot it further. 😊

# User

See here:

aging\_list = [

(lambda day, amount: amount < 0, "Advance"),

(lambda day, amount: 0 <= day < 31, "Not Due"),

(lambda day, amount: 31 <= day < 61, "31-60 Days"),

(lambda day, amount: 61 <= day < 91, "61-90 Days"),

(lambda day, amount: 91 <= day < 181, "91-180 Days"),

(lambda day, amount: 181 <= day < 366, "6 Months to 1 year"),

(lambda day, amount: 366 <= day, "More than 1 year")

]

You have not converted variable day and amount into integers?

# Bing

You're correct. In the aging\_list where you're defining your lambda functions, the variables day and amount are not explicitly converted to integers. However, this should not be a problem if they are already integers when the lambda functions are called.

In your code, you've converted 'days' and 'amount' to integers before creating the 'days\_amounts' column:

df['days'] = df['days'].astype(int)

df['amount'] = df['amount'].astype(int)

df['days\_amounts'] = list(zip(df['days'], df['amount']))

So, when you're applying the lambda functions in aging\_list to the 'days\_amounts' column, day and amount should already be integers.

The error message you're seeing suggests that 'days' and 'amount' are being treated as strings during the comparison operation in your map function. This could be due to the data type of these fields not being properly converted to integers.

If you're still facing issues, you could try adding explicit integer conversions in your lambda functions, like so:

aging\_list = [

(lambda day, amount: int(amount) < 0, "Advance"),

(lambda day, amount: 0 <= int(day) < 31, "Not Due"),

(lambda day, amount: 31 <= int(day) < 61, "31-60 Days"),

(lambda day, amount: 61 <= int(day) < 91, "61-90 Days"),

(lambda day, amount: 91 <= int(day) <

# User

can you regenerate above response?

# Bing

Certainly, here's the information again:

You're correct. In the aging\_list where you're defining your lambda functions, the variables day and amount are not explicitly converted to integers. However, this should not be a problem if they are already integers when the lambda functions are called.

In your code, you've converted 'days' and 'amount' to integers before creating the 'days\_amounts' column:

df['days'] = df['days'].astype(int)

df['amount'] = df['amount'].astype(int)

df['days\_amounts'] = list(zip(df['days'], df['amount']))

So, when you're applying the lambda functions in aging\_list to the 'days\_amounts' column, day and amount should already be integers.

The error message you're seeing suggests that 'days' and 'amount' are being treated as strings during the comparison operation in your map function. This could be due to the data type of these fields not being properly converted to integers.

If you're still facing issues, you could try adding explicit integer conversions in your lambda functions, like so:

aging\_list = [

(lambda day, amount: int(amount) < 0, "Advance"),

(lambda day, amount: 0 <= int(day) < 31, "Not Due"),

(lambda day, amount: 31 <= int(day) < 61, "31-60 Days"),

(lambda day, amount: 61 <= int(day) < 91, "61-90 Days"),

(lambda day, amount: 91 <= int(day) < 181, "91-180 Days"),

(lambda day, amount: 181 <= int(day) < 366, "6 Months to 1 year"),

(lambda day, amount: 366 <= int(day), "More than 1 year")

]

I hope this helps! Let me know if you have any other questions. 😊

# User

After above change there is another error:

File "lib.pyx", line 2972, in pandas.\_libs.lib.map\_infer File "D:\OneDrive - 0yt2k\excel examples including add-in xlwings and connected python - vba - dlls, etc\xlwings\_trial.py", line 81, in <lambda> ddf['age\_bucket'] = ddf['days\_amounts'].map(lambda x: next((label for condition, label in aging\_list if condition(x[0], x[1])), None), meta=('days', 'object')) ^^^^^^^^^^^^^^^^^ File "D:\OneDrive - 0yt2k\excel examples including add-in xlwings and connected python - vba - dlls, etc\xlwings\_trial.py", line 81, in <genexpr> ddf['age\_bucket'] = ddf['days\_amounts'].map(lambda x: next((label for condition, label in aging\_list if condition(x[0], x[1])), None), meta=('days', 'object')) ^^^^^^^^^^^^^^^^^ File "D:\OneDrive - 0yt2k\excel examples including add-in xlwings and connected python - vba - dlls, etc\xlwings\_trial.py", line 74, in <lambda> (lambda day, amount: 0 <= int(day) < 31, "Not Due"), ^^^^^^^^^^^^^^^^^ ValueError: invalid literal for int() with base 10: '('

# Bing

The error message ValueError: invalid literal for int() with base 10: '(' indicates that the program is trying to convert the string '(' to an integer, which is not possible and hence, it's throwing an error.

Looking at your code, it seems like the issue might be with how the data is being read into the DataFrame. The error suggests that at some point, a tuple is being read as a string, hence when you try to access the elements of the tuple (which are supposed to be integers), you're actually accessing the characters of the string.

Here's a small example to illustrate this:

x = '(694, 213072)' # This is a string, not a tuple

print(x[0]) # This will output '('

In your case, it seems like x in the lambda function lambda x: next((label for condition, label in aging\_list if condition(x[0], x[1])), None) is a string that looks like a tuple, rather than an actual tuple of integers.

To fix this, you need to ensure that 'days\_amounts' is a list of tuples, not a list of strings. You can do this by modifying the line where you create 'days\_amounts':

df['days\_amounts'] = list(zip(df['days'], df['amount']))

to:

df['days\_amounts'] = [eval(i) for i in df['days\_amounts']]

The eval() function evaluates a string as a Python expression, so it will convert strings that look like tuples into actual tuples.

Please try this modification and let me know if it helps! 😊

# User

See I think the basic issue is you are considering inputs of days and amount as list instead of list of list. Could that be problem?

# Bing

Yes, you're correct. The days\_amounts is expected to be a list of lists (or a list of tuples), where each inner list (or tuple) contains two elements: 'days' and 'amount'. If 'days' and 'amount' are passed as separate lists instead of a list of lists (or tuples), it could certainly cause issues.

In your original code, you're creating 'days\_amounts' as a list of tuples:

df['days\_amounts'] = list(zip(df['days'], df['amount']))

This is the correct approach if 'days' and 'amount' are separate lists. The zip function pairs the corresponding elements from the 'days' and 'amount' lists together into tuples, and list(zip(...)) converts these pairs into a list of tuples.

However, if 'days' and 'amount' are already paired together in a list of lists (or tuples), you don't need to use zip or list(zip(...)). You can simply assign 'days\_amounts' directly:

df['days\_amounts'] = days\_amounts

I hope this helps! Let me know if you have any other questions. 😊

# User

I think the issue seems to be there only. See the below code similar but uses pandas:

@xw.func @xw.arg('days\_amounts', ndim=2) def aging\_buckets(days\_amounts): aging\_list = [ (lambda day, amount: amount < 0, "Advance"), (lambda day, amount: 0 <= day < 31, "Not Due"), (lambda day, amount: 31 <= day < 61, "31-60 Days"), (lambda day, amount: 61 <= day < 91, "61-90 Days"), (lambda day, amount: 91 <= day < 181, "91-180 Days"), (lambda day, amount: 181 <= day < 366, "6 Months to 1 year"), (lambda day, amount: 366 <= day, "More than 1 year") ] result = [[next((label for condition, label in aging\_list if condition(day[0], day[1])), 'No Match')] for day in days\_amounts if len(day) == 2] return result It runs perfactly. I think: df['days\_amounts'] = list(zip(df['days'], df['amount'])) may not be necessary even if you use dask?

# Bing

You're correct. If days\_amounts is already a list of lists (or tuples) where each inner list (or tuple) contains two elements ('days' and 'amount'), then you don't need to use zip to pair them together.

In the pandas version of your function, days\_amounts is directly passed as an argument, so it's already in the correct format. However, in the Dask version of your function, you're creating a DataFrame from days\_amounts, and then trying to recreate days\_amounts from the DataFrame. This is where the issue arises.

If days\_amounts is already in the correct format, you can directly create a Dask DataFrame from it without needing to zip 'days' and 'amount' together. Here's how you can do it:

ddf = dd.from\_pandas(pd.DataFrame(days\_amounts, columns=['days', 'amount']), npartitions=4)

In this line, a pandas DataFrame is created from days\_amounts, and then a Dask DataFrame is created from the pandas DataFrame. The resulting Dask DataFrame will have two columns, 'days' and 'amount', which correspond to the elements of the inner lists (or tuples) in days\_amounts.

I hope this helps! Let me know if you have any other questions. 😊

# User

But instead of npartitions my code uses client:

@xw.func @xw.arg('days\_amounts', ndim=2) def aging\_buckets\_parallel(days\_amounts): client = Client(n\_workers=4, threads\_per\_worker=2) df = pd.DataFrame(days\_amounts, columns=['days', 'amount']) df['days'] = pd.to\_numeric(df['days'], errors='coerce') df['amount'] = pd.to\_numeric(df['amount'], errors='coerce') df['days\_amounts'] = list(zip(df['days'], df['amount'])) ddf = dd.from\_pandas(df, npartitions=4) aging\_list = [ (lambda day, amount: int(amount) < 0, "Advance"), (lambda day, amount: 0 <= int(day) < 31, "Not Due"), (lambda day, amount: 31 <= int(day) < 61, "31-60 Days"), (lambda day, amount: 61 <= int(day) < 91, "61-90 Days"), (lambda day, amount: 91 <= int(day) < 181, "91-180 Days"), (lambda day, amount: 181 <= int(day) < 366, "6 Months to 1 year"), (lambda day, amount: 366 <= int(day), "More than 1 year") ] ddf['age\_bucket'] = ddf['days\_amounts'].map(lambda x: next((label for condition, label in aging\_list if condition(x[0], x[1])), None), meta=('days', 'object')) result\_df = ddf.compute() result = [[item] for item in result\_df['age\_bucket'].values] client.close() return result

# Bing

Yes, you're correct. The Client object in Dask allows you to connect to a Dask distributed scheduler which you've started with Client(n\_workers=4, threads\_per\_worker=2). This starts a local Dask cluster for you on your machine.

The npartitions argument in dd.from\_pandas(df, npartitions=4) is used to specify the number of partitions you want for your Dask DataFrame. It's not related to the Client object but is a parameter for how you want to partition your data in Dask.

Regarding the issue with your code, the line df['days\_amounts'] = list(zip(df['days'], df['amount'])) is indeed not necessary if days\_amounts is already a list of lists where each inner list contains two elements: 'days' and 'amount'.

In fact, this line could be causing the issue because zip(df['days'], df['amount']) will create a list of tuples, and when this is assigned to a DataFrame column, the tuples are converted to strings. This is why you're seeing a string representation of a tuple in your error message.

You can remove this line and directly use 'days' and 'amount' in your DataFrame. Here's how you can modify your code:

@xw.func

@xw.arg('days\_amounts', ndim=2)

def aging\_buckets\_parallel(days\_amounts):

client = Client(n\_workers=4, threads\_per\_worker=2)

df = pd.DataFrame(days\_amounts, columns=['days', 'amount'])

df['days'] = pd.to\_numeric(df['days'], errors='coerce')

df['amount'] = pd.to\_numeric(df['amount'], errors='coerce')

ddf = dd.from\_pandas(df, npartitions=4)

aging\_list = [

(lambda day, amount: int(amount) < 0, "Advance"),

(lambda day, amount: 0 <= int(day) < 31, "Not Due"),

(lambda day, amount: 31 <= int(day) < 61, "31-60 Days"),

(lambda day, amount: 61 <= int(day) < 91, "61-90 Days"),

(lambda day, amount: 91 <= int(day) < 181, "91-180 Days"),

(lambda day, amount: 181 <= int(day) < 366, "6 Months to 1 year"),

(lambda day, amount: 366 <= int(day), "More than 1 year")

]

ddf['age\_bucket'] = ddf.map\_partitions(lambda df: df.apply(lambda row: next((label for condition, label in aging\_list if condition(row['days'], row['amount'])), None), axis=1), meta=('days', 'object'))

result\_df = ddf.compute()

result = [[item] for item in result\_df['age\_bucket'].values]

client.close()

return result

In this modified code, I've removed the line df['days\_amounts'] = list(zip(df['days'], df['amount'])) and changed the map function to map\_partitions with a lambda function that applies the aging conditions to each row of the DataFrame.

I hope this helps! Let me know if you have any other questions. 😊

# User

You have used:

client = Client(n\_workers=4, threads\_per\_worker=2) and finally client.close()

But in the code, I do not find their use elsewhere. Does npartitions parameter not achieve parallel processing without client? What is the role of those Client commands in speeding up the code?

# Bing

The Client object in Dask is used to set up a distributed computing environment. When you create a Client, you're starting a local Dask cluster on your machine. The parameters n\_workers and threads\_per\_worker specify the number of workers and threads per worker in this cluster.

The npartitions parameter in dd.from\_pandas(df, npartitions=4) is used to partition your data into chunks. Each chunk is a separate pandas DataFrame that can be processed independently of the others. This allows Dask to perform computations in parallel, with each worker in the cluster processing a different partition.

However, simply partitioning your data with npartitions does not automatically set up a distributed computing environment. This is where the Client comes in. By creating a Client, you're telling Dask to distribute the computations across the workers in your cluster.

So, while npartitions allows for parallel processing by breaking your data into chunks, the Client is what actually sets up the parallel computing environment.

Finally, client.close() is used to shut down the cluster once you're done with it. This is good practice because it frees up system resources.

In summary, both Client and npartitions play important roles in enabling parallel processing in Dask. The Client sets up the distributed computing environment, and npartitions allows your data to be processed in parallel by breaking it into chunks. 😊

# User

But where int he code, partitions are distributed to the client. There does not seem to be any command for that. Or that is done under the hood?

# Bing

Yes, the distribution of partitions across the workers in the Dask cluster is handled under the hood when you call ddf.compute().

When you create a Dask DataFrame, it's essentially a large logical DataFrame composed of many smaller pandas DataFrames split along the index. These smaller pandas DataFrames (partitions) can be computed on different threads, processes, or even machines.

When you call ddf.compute(), Dask automatically schedules the computations on these partitions across the available workers in the cluster. It handles the distribution of data and the scheduling of tasks.

So, you don't need to manually distribute the partitions to the workers. Dask takes care of this for you when you execute your computation. 😊

# User

What is meant by cluster? Is it full dataframe or what?

# Bing

In the context of Dask and distributed computing, a "cluster" refers to a group of machines (or cores on a single machine) that work together to perform computations.

When you create a Client in Dask, you're starting a local cluster on your machine. This cluster can consist of multiple workers, each of which can process tasks independently of the others. If you're running on a multi-core machine, each worker can run on a separate core, allowing for parallel processing.

The term "cluster" does not refer to the full DataFrame or any part of the data itself. Instead, it refers to the computing resources (machines or cores) that are used to process the data.

When you call ddf.compute(), Dask automatically schedules the computations on the partitions of your DataFrame across the available workers in the cluster. This is how Dask enables parallel processing and handles larger-than-memory computations. 😊

# User

What is the role of threads because suppose I have 4 cores, there will be 4 clients and each will handle processing. Does one core handle multiple threads?

# Bing

In the context of computing, a thread is the smallest unit of execution that can be scheduled by an operating system. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources.

When you create a Client in Dask with n\_workers=4, threads\_per\_worker=2, you're starting a local cluster on your machine with 4 workers, each of which can run 2 threads concurrently. This doesn't necessarily mean you have 4 cores, it's just how you've chosen to set up your Dask cluster.

If your machine has multiple cores, then yes, each core can handle multiple threads. The operating system's scheduler is responsible for assigning threads to different cores and managing their execution. This allows for true parallel processing, where multiple threads can be executed at the same time on different cores.

However, even on a single-core machine, it can appear as though multiple threads are being executed at the same time. This is because the operating system rapidly switches between threads in a process known as context switching. But in reality, only one thread can be executed at a time on a single core.

So, in your Dask setup, if you have a multi-core machine, you could potentially have multiple threads running concurrently on each core. The exact number would depend on how the operating system schedules the threads. 😊