Dask

Kangdong yuan

**Introduction**

Dask is an open-source library for parallel computing written in Python. Dask compose of two parts. It includes a task scheduling component for building dependency graphs and scheduling tasks. Second, it includes the distributed data structures with APIs similar to Pandas Data frames or NumPy arrays. Moreover, the Dask can provide running time analysis for paralleled program by tunning the parameters.

In this document, I will show the feature and application of Dask. First step is implementing the paralleled program by Dask. Second, I will generate the graph of process of paralleled computation by built-in function in Dask. Third, I will change the number of workers (cores) and threads of each workers and analysis the running time. Finally, I will give suggestions for the Dask in different usage scenarios.

**Section 1: The feature of Dask**

The Dask library is an open-source library that help people to do parallel computing. And many built-in features of dask enable it can do the paralleling computing with precision and efficiency.

The first important feature of dask is it can compatible with many data type of python such as NumPy, pandas, scikit-learn. So dask is really easy and convenient to use in python environment. Especially for large scale data computation such as matrix multiplication, database setup and machine learning.

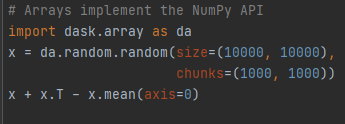


Figure 1 matrix multiplication by dask and NumPy

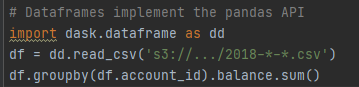


Figure 2 data frame setup by dask-pandas

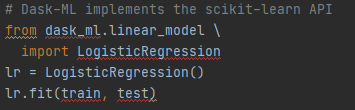


Figure 3 machine learning by dask-sklearn

The second important feature of dask is that it only plans the computation process when we run the program. If we need to get the result, we need to use the computation function to tell the program to start execution. In the dask, this computation feature is called delayed feature. So, in program, the variables in computation are not real data, and the variables are delayed objects. However, we can treat those delayed objects as variables, and those delayed objects will not be computed until we ask program to do it. This delayed object feature is the idea that has been used in spark for distributed and paralleled computation. There are two reason for dask to use delayed feature. First, when some people only need to do analysis of algorithm, they just need the schedule of tasks of algorithm without actual computation. So, delayed feature enables people to do analysis of program without waste time on computation. Second, delayed feature enable program to do computation on distributed machines, so dask need define a running plan first, then each task will be sent to other machine to calculate.

Finally, those two features of dask were designed for do various tasks (math computation, database, machine learning) by distributed and paralleled computation.

**Section 2: The** **visualization of procedure of multi-threading process**

If we design a paralleled algorithm, it often causes confusion to us when we analysis the time complexity and the actual process of computation. For example, if we want to get the summation of the elements in a list by prefix sum, we need add each element by the branch factor n, and the whole computation process in idle situation is a tree with n child nodes for each internal node. In dask, it offers a convenient and special way to make the process of this computation into a graph.

This graph is the most basic component of visualization. The “increment” is the name of function we applied to the data, and the arrow mean the data have been processed by increment function, and the rectangle represent the data after transformation.

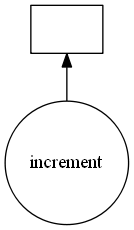


Figure 4 basic component of visualization

If we want to make the graph of algorithm, we just need to replace the variables with delayed objects and return result as delayed objects. After we get the final delayed object, we can use “delayed\_object.visualize("system-path/graph.png")” to generate the graph on local machine.

The graph below is the visualization is the prefix sum by branch factor = 3. So, we can see that child nodes of each internal node don’t exceed by 3. And, this graph is a perfect tree structure that is same as our hand-written graph of prefix sum. This graph can help we a lot when we need to analysis time complexity, because this tree structure enables us to calculate time complexity directly. Moreover, it proves the correctness of our program, since it shows that our algorithm compute by tree structure.

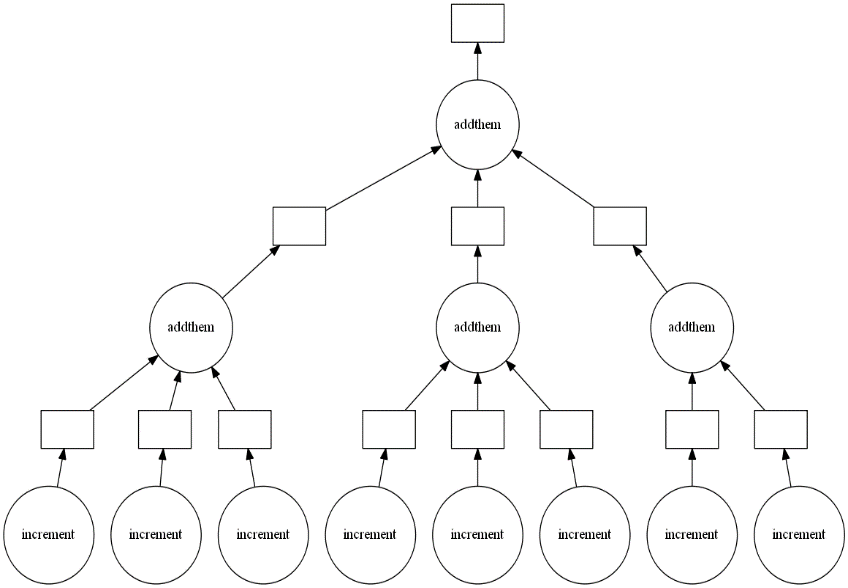


Figure 5 visualization of prefix sum by factor 3

Besides analyzing and proving the algorithm, we also can use dask to make visualize. And, the advantage of dask visualization is that it can use many python libraries such as NumPy, pandas. So, it is convenient for us to do math computation.

For example, we do the matrix transpose and matrix addition by NumPy. First step is using import the dask array. The second step is generating the matrix by NumPy array function, and this matrix has been divided by 5. Then, I print this matrix by using “compute()” function to retrieve the result.

Fourth, I do the matrix transpose and matrix addition and use y as delayed object that store the final result. Finally, I save the graph as “transpose.png”.

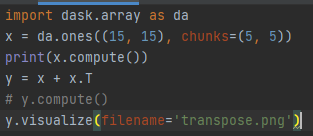


Figure 6 perform the matrix calculation

Then, the generating graph is below. The graph shows clear path and detailed parameters for each step. From this graph, we can see the matrix has been transposed or two matrices have been added together. Besides this simple matrix calculation, dask also can make very complex graph to represent some complex math calculation or algorithm.

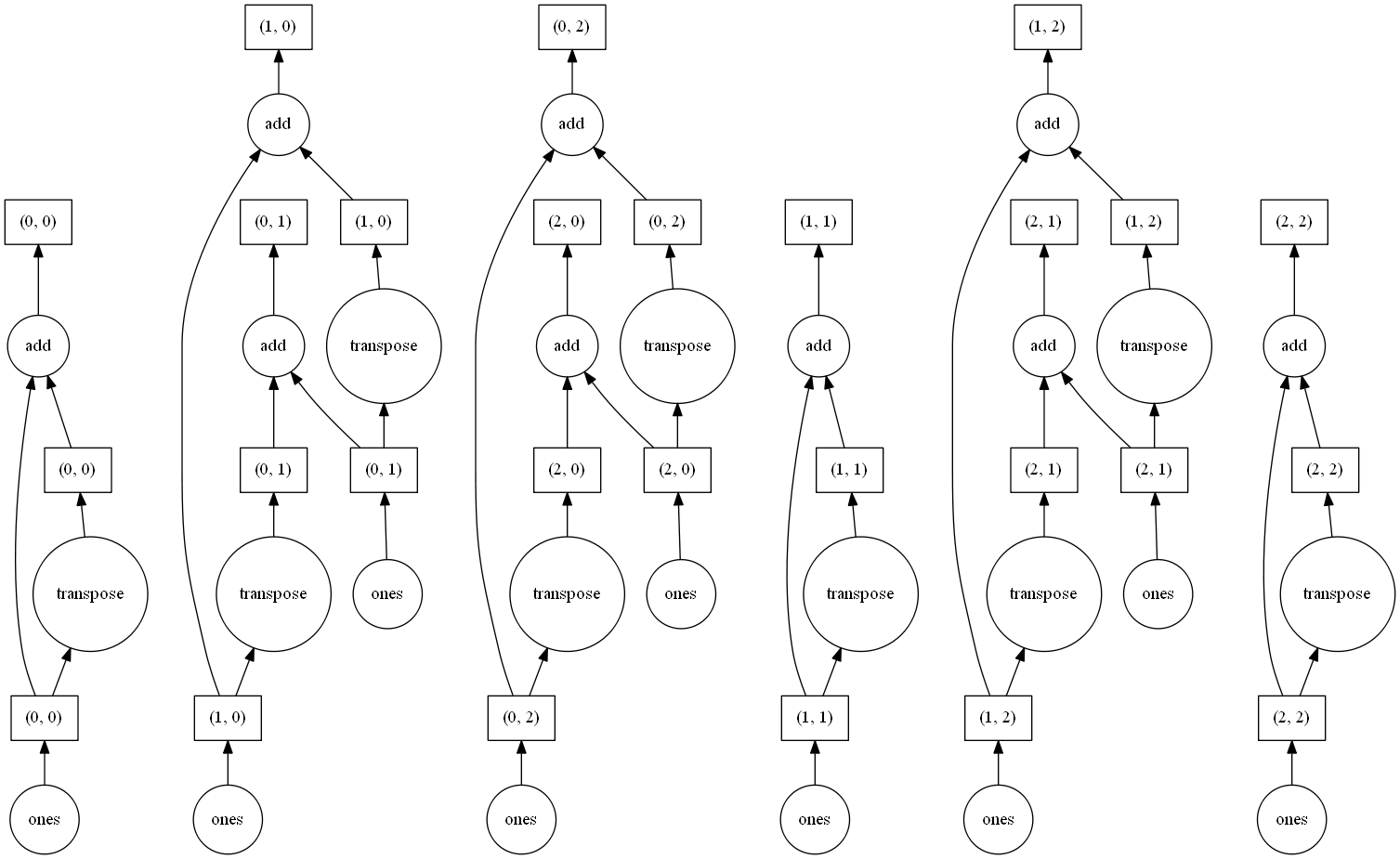


Figure 7 visualization of matrix calculation

Moreover, if we need to plan and manage a complex and large-scale program. We need to have an intuitive understanding of the program flow and running sequence. The visualization equation of dask can visualize the running program. First, dask will build a flowchart of the program algorithm structure, and then when we run the program, the various steps in the flowchart will light up in turn to remind us which part of the program is currently running.

The flowchart of dask can be used when a single machine is running, or when multiple machines are running in a distributed manner. This visualization function allows us to visually see the sequence and process of program execution, as well as how a distributed system runs a program in parallel. Finally, this animated picture can be saved to the local machine for us to analyze the algorithm.

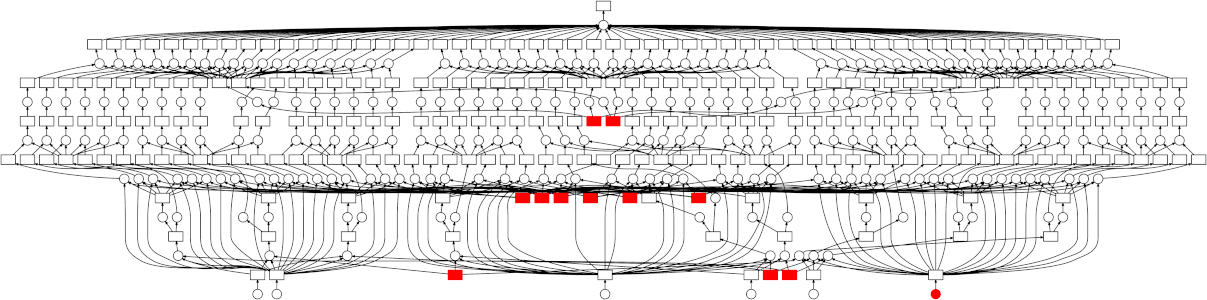


Figure 8 The gif of program running (dask)

Through the demonstration of these examples, we can see that the visualization function of dask is very powerful and can be applied in different environments and data structures. If we need to use pictures to analyze and explain certain programs and algorithms, the visualization function of dask is a very helpful way. But the visualization function of dask also has some shortcomings. First, we cannot customize the image style. In the visualization of dask, the function is always a circle and the data is always a rectangle. Second, when we need to visualize a very complex program, the picture will become too large, so that the picture browser cannot open such a huge picture.

**Section 3: The performance of different number of worker and thread**

As a library that supports parallel operation, dask can define the number of threads in the runtime environment. There are two parameters in the multi-threaded environment of dask. The first is the number of workers, which represents how many cores the program runs at the same time. The second parameter is the number of threads for each worker, which represents how many things each worker can do at the same time.

To measure the performance, I choose the prefix sum as the testing program and set the branch factor to 2. And, my test environment is windows10; CPU is AMD 4800H (Clock speed: 2.9 GHz, Cores: 8, Threads: 16); L1 cache is 512kb, L2 cache is 4mb, and L3 cache is 8mb.

And I need to mention that I need convert my prefix sum function to multi-threading version. In this version, the visualization function cannot be used.

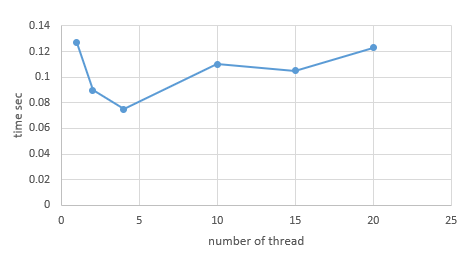


Figure 9 performance on L1 cache level

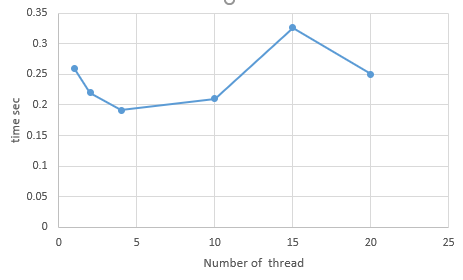


Figure 10 performance on L2 cache level

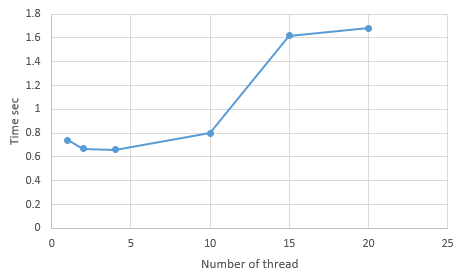


Figure 11 performance on L3 cache level

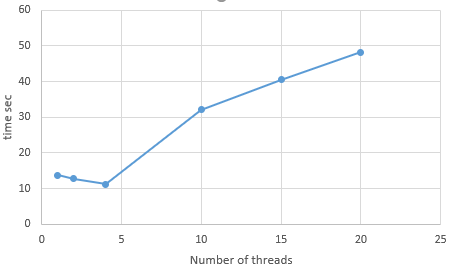


Figure 11 performance on memory cache level

As can be seen from the above chart, the running time of the program shows a very strange pattern and does not meet our expectations. First of all, we expect that as the number of threads (the number of workers) increases, the running time of the program will become shorter. But the performance of the program at every memory level does not meet our predictions. Second, we believe that the running time will not fluctuate as the number of threads changes. But we see that the running time of the program in the chart is very unstable and does not follow any rules.

For these problems encountered in running programs, I consulted the official dask documents and user communities to find the answers. First, in the official dask document, the developer said that dask is an open-source library for users to analyze and test parallel and distributed programs. But dask does not guarantee that it will improve the efficiency of the program. Second, dask is generally used to run on distributed systems, and I use my laptop to run these programs. When I use more threads to run my program, maybe my computer does not meet the ideal operating environment of dask. So, when I increase the number of threads, my program runs longer. Third, my program uses python's while loop to dispatch tasks. Although dask can run in parallel, the while loop has a linear running time. So, when I have more threads, my program has used linear time to dispatch tasks.

**Section 4: The usage case and potential application**

Dask is a very good tool to help a beginner learn how to design parallel and distributed programs. First, dask is built on the python environment, which makes it very easy to learn and easy to operate. Second, dask can use many data structures, including matrices, graphs, linked lists, and binary trees, which allows dask to do many mathematical and statistical topics. Third, dask runs and calls many classic machine learning algorithms, which makes it easy for users to deploy machine learning on a distributed system. Fourth, dask has a very simple visualization program that allows users to generate concise pictures to explain the calculation process.

But dask also has some shortcomings. First, dask is not a very efficient program, so using dask to implement parallelism does not necessarily guarantee the efficiency of the program. Second, the visualization of dask is limited to small-scale programs, and there is no option for users to customize.

So, I recommend using dask for teaching, theoretical verification and conference presentations. These scenarios do not require a huge amount of calculation, so the efficiency of the program is not very important. And for theoretical verification and teaching, some simple visualization pictures are enough. And students can learn to use dask technology without much time.

**Citation list**

group, dask. (n.d.). Scheduling. Dask documentation. <https://docs.dask.org/en/latest/scheduling.html>.