RU Jupyterhub with IPyParallel

PHPC - Project

Sigurður Baldursson
Reykjavík University
sigurdur14@ru.is

May 20, 2023

Keywords: IPython, Clusters, IPyParallel & Processes

Executive Summary

The IPython (Interactive Python) interpreter first came forth in 2001 [1] and remained a console shell tool for years. Until 2011 the first release of a notebook emerged and in 2014 it became Jupyter Notebook which is commonly used for interactive computing in Python or other languages for programming, ML, data science, econometrics, sharing data stories and research results.

In IPyparallel we have an IPcluster, that starts a connection, a client so a user can interact with the IPengines. [2] The IPcluster runs one IPcontroller that can have many IPengines, that are the individual processes that can be instructed to run tasks separately or in parallel.

At it's base layer IPython is a messaging protocol. Any language with ZMQ bindings (about 40 languages), can do an IPython kernel implementation. The only difference between an engine in IPyparallel and a IPython kernel is that an engine executes while a kernel listens for commands. This report shows that you can set up an IPython cluster configuration in a users IPython profile on Jupyterhub, and a user can start a cluster from the UI. Some applications of IPyparallel on Jupyterhub would be to to make Jupyter notebooks assignments for distributed training and hyper-parameter optimization more efficient or use it as a teaching exercise in MPI. For the project I implemented a connection to Jötunn, a HPC cluster operated by Reiknistofnun Háskóla Íslands that is SSH accessible from the Reykjavik University network.

1 Background

Single Instruction, Single Data (SISD) programs are often run on a small scale computing systems such as a laptop but as more and more students start to learn about machine and deep learning the size and complexity of computation increases with it and scaling is needed. Programs that run for minutes instead of hours or days on a regular laptop need to be moved to a parallel execution by connecting them to a cluster that is a large scale computing system with multiple nodes, CPU's and GPU's.

According to a short research by Kim et al. [3] they compared processing times between a serial run on one core versus IPyParallel using 6 engines (Presuming that the authors of this paper mean that each engine had access to the same 6 CPU cores as are listed in the hardware environment, as number of IPengines does not mean the same as the number of CPU cores on a machine) They also compared Pandaral-lel library, mainly used for parallel tasks for the Pandas library, and compared 1 core vs 6 cores. They did not use the same algorithm for the Pandaralel and IPyparallel, but IPyparallel showed a 62.3% performance improvement from a serial execution and Pandaral-lel improved it's pandas algorithm by 80%.

There are many libraries in Python made for different distributed and parallel tasks, such as Dask, Bodo, Fiber, Ray, Dispy, Pandaral-lel, PySpark and Joblib. Dask, Bodo and Fiber are the more general-purpose parallel computing libraries, while IPyparallel is tightly integrated with the Jupyter ecosystem. That is why IPyparallel has integrated the possibility of using Dask [4] which gives it the flexibility of using the high level Dask API for different data structures that Dask provides. With IPyparallel defining the current improvements

in improving interactivity and being a good case for most parallel tasks but excelling above other tools in enabling interactivity for traditional single program, multiple data (SPMD), e.g. MPI workloads, particularly during prototyping and debugging stages. For those who want to run an MPI simulation and desire the ability to pause it midway, explore the data, generate plots, and interact with individual or all nodes and leveraging the interactive capabilities offered by Jupyter, then IPyparallel might be the ideal tool. [5]

When IPengines registers with the controller for the same cluster id, the IPcontroller publishes a message to the IPcluster that says, I just got a new engine, so the IPcluster is always up to date with the current situation and can dynamically add new engines. When code messages are sent from the users notebook the Scheduler receives it and schedules all actions that go to the engine. Engines are stateful and each engines is it's own process. I will not go into describing all the documentation but mostly the important parts of parallelism.

1.1 Views: Styles of Parallelism

There are three types of Views that the Client can use to interface with, to execute code on engines. The Direct interface where engines are address explicitly and you can make them run in parallel or individually. The Direct view has a time complexity of $O(N)$ when addressing all engines, mainly because it is implemented on the client side by a loop that sends messages to engines individually. The Loadbalanced interface where the scheduler is trusted to assign work to appropriate engines. The newest style of parallelism in IPyparallel is called the Broadcast View and is enabled by the new experimental broadcaster in IPython 7 (Jötunn has IPython 6.0.0), that only sends one message to the Scheduler, that handles the fan-out to engines and has a time complexity of $O(1)$. [6]

2 Methodology

I set up IPyParallel on Jupyterhub. A more technical description can be found in the accompanying github repository [7] but the high level setup is as follows:

The IPcluster is controlled from Jupyterhub, with the IPcontroller and IPengines on the cluster. To have the IPcontroller and IPengines on the same server minimizes the communication overhead since the controller can send messages to multiple IPengines while the IPcluster to IPcontroller is a one on one communication.

For the setup it is relatively minimal and admin friendly, as most of the configuration is set in one configuration file on Jupyterhub so no user specific .ipython or ipyparallel pre-configuration is required on the Jötunn cluster machines. In general this means that each user account on the cluster does not need to have a preconfigured ipyparallel profile in the .ipython folder and does not need any .ipython folder on the cluster even. But on Jupyterhub each user should have the jotunn profile in his user .ipython folder. The only thing needed on the cluster is to have a bash login script on the remote cluster that can be loaded by the users. The script loads the required modules and calls python and sets up the IPython profile with information from the configuration from Jupyterhub. The IPcluster configuration file on Jupyterhub tells it to start the controller and engines with sshproxy and MPI, i.e. it will run from Jupyterhub and ssh into jotunn.rhi.hi.is, execute the login bash shell that loads the users environment from bashrc and profile and runs the script that loads modules first, then runs ipcluster engines from profile jotunn with mpi, which will amount to mpiexec ipengine on Jötunn. That is also why we load the required modules for Open MPI in the script.

For the setup it is given that each student has the same Canvas student login_id username (not password) on the Jötunn cluster as on Jupyterhub. When Jupyterhub spawns a single-user server for a student with the Jotunn profile already in a students .ipython directory, The student only needs to setup their ssh keys on Jupyterhub and then all they have to do is start the cluster from the IPython Clusters tab by selecting the number of engines and pressing start. More advanced users can use the ipcluster terminal command or do it from the notebook. When starting a cluster, we wait around 10-30 seconds or until we see the message that "Engines appear to have started successfully" (The output from starting ipcluster in a terminal is shown in detail in the accompanying Github repo [7]). Otherwise it might get a timeout if connecting with the client from the notebook to fast.

3 Results

Having the setup like this proved successful. Having configuration mostly on the local side minimises the setup to only one version of the ipython profile configuration. I also found out when shutting down a user notebook server on Jupyterhub, that IPyparallel also has a teardown method that it uses to shut down the cluster controller and engines when it stops. The connection seems to be very stable as a static notebook server

that I also have on Jupyterhub. I could connect to the client even the next day where as regular user servers are culled after one hour of inactivity. Jötunn would allow me to start 16 engines at most, with mpiexec saying there are not enough slots available in the system to satisfy the 17 slots requested, but IPyParallel can support up to 10 thousand engines.

Due to some package inconsistencies on Jötunn some methods in ipyparallel did not work. The serializer, for example ipyparallel deserialization methods must be the same as the one that serialized it. But the pip on Jötunn could not find the same package. Updating pip and the default IPython on Jötunn just resulted in other errors. I was not able to install some old packages to match the Jötunn environment. Some errors from this are described under "Current issues" in the accompanying Github repo [7]. This repository also has detailed setup with code samples and a demo notebook with parallel code execution and MPI sample.

4 Future work

Ideally the Python that is run on the cluster and Jupyterhub should be the same or at least be able to run the same version of IPython IPyparallel since it uses serialization methods and for moving code and messaging protocols that have to match.

Next steps would be to try to setup Jötunn and Smallvoice with Slurm. The only problem is that I don't have a VPN set up yet on Jupyterhub to be able to SSH into the same network as Smallvoice. (Asking UTS at Reykjavik University it is possible to move Jupyterhub to that network). One idea for running the engines on separate compute nodes on the clusters may be to get Slurm to launch the engine on the compute nodes by targeting each compute node in Slurm with $-w = / - -nodelist =$ nodename/s. That way it seems that each engine could be started on each node but I am not sure of it would work as expected. I tested also logging into compute nodes on Smallvoice and I got an importerror for the ipengine command on plato for example.

For IPyParallel you can define a Slurm batch script and it will always have {n} and {profile_dir} variables passed to the formatter so that it may be possible to divide those engines that a user wants to start on different compute node machines.

5 Discussion

Despite having some troubles setting this up on the current RU-Jupyterhub instance as it had a mix of packages on pip and conda making installing ipyparallel have duplicate packages. I had to move everything to the conda side and install IPyparallel and the jupyter_server packages. Playing around with MPI and learning more about parallel execution was very interesting.

The future of Jupyterhub remains unclear at Reykjavik University as I have not been advertising it since 2019, but the teachers in Data Science keep asking for it so I keep on updating it and users seem to be very active. It would be cool to take it to the next level and run it in a more safe way, maybe on Kubernetes. With the latest notebook assignment and file exchange (nbexchange) having recently been implemented we could run it as we have for assignments and grading also. With more computing power and parallel execution students can learn how to execute in parallel and learn MPI. The notebook grader can also be used as a file sharing service, to send whatever code is needed to get started.

References

[1] *Project Jupyter*, en, Page Version ID: 1148865829, Apr. 2023. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Project_Jupyter&oldid=1148865829` (visited on 05/02/2023).

[2] IPython, *Ipyparallel*, 2015. [Online]. Available: `https://github.com/ipython/ipyparallel`.

[3] T. Kim, Y. Cha, B. Shin, and B. Cha, "Survey and Performance Test of Python-based Libraries for Parallel Processing," in *The 9th International Conference on Smart Media and Applications*, ser. SMA 2020, New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 154–157, ISBN: 978-1-4503-8925-9. DOI: `10.1145/3426020.3426057`. [Online]. Available: `https://dl.acm.org/doi/10.1145/3426020.3426057` (visited on 05/17/2023).

[4] *Working with IPython and dask.distributed — ipyparallel 8.6.2.dev documentation*. [Online]. Available: `https://ipyparallel.readthedocs.io/en/latest/examples/dask.html` (visited on 05/19/2023).

[5] M. RK, *IPython Parallel in 2021*, en, Nov. 2021. [Online]. Available: `https://blog.jupyter.org/ipython-parallel-in-2021-2945985c032a` (visited on 05/19/2023).

[6] *Broadcast View — ipyparallel 8.6.2.dev documentation*. [Online]. Available: https://ipyparallel.readthedocs.io/en/latest/examples/broadcast/Broadcast%20view.html (visited on 05/19/2023).

[7] S. Baldursson, *Jupyterhub ipyparallel*, https://github.com/sigurdurb/jupyterhub-ipyparallel, Accessed: May 20, 2023, 2023.

Appendix A. Use of ChatGPT

At first I asked OpenAI ChatGPT general questions on how to set up a IPyParallel cluster with TCP connection or similar. But I could see from the naming convention of objects and methods, such as HubFactory and IPEngineApp that can't be found in the latest documentation on https://ipyparallel.readthedocs.io/en/latest/ that it was using an old version of the documentation. So I mostly stuck to reading the documentation and the Github code. I also asked it some questions about how Dask and Fiber differentiate and compare it to IPyParallel but also found a more reliable distinction in a 2021 article on the Jupyter blog that goes over the most important developments over recent years and the future trajectory of the project: https://blog.jupyter.org/ipython-parallel-in-2021-2945985c032a