Module 6: Critical Thinking

Ryan Thompson

Colorado State University - Global

CSC 507

Dr. Farr

27 November 2024

Python File Processing

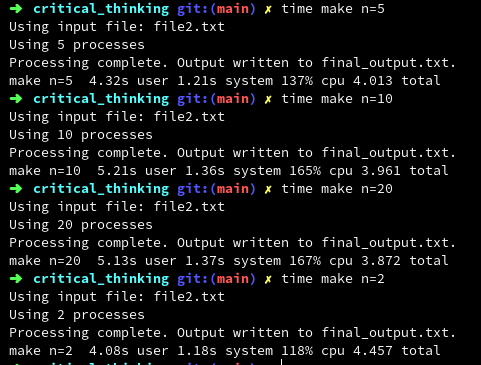
Efficient data processing is a critical factor in handling large datasets, especially in fields like data science and machine learning. This paper evaluates the impact of different methods for splitting a large input file into smaller parts and processing them in parallel using Python. Specifically, we will examine various splitting strategies and their associated processing times, focusing on the most effective approach to speed up the process. The analysis is based on the results of an experiment where the input file was divided into 2, 5, 10, and 20 smaller parts, with processing performed in parallel.

The experiment aimed to determine the most efficient method for processing a large file by splitting it into multiple parts. After splitting, each part was processed in parallel, and the resulting files were combined into a single output file. The processing times for each method were recorded to identify the fastest approach. Results showed that splitting the file into 10 or 20 parts produced similar processing times, which were significantly faster than splitting the file into 2 parts. The 5-part method was faster than the 2 part method but slower than the 10 part and 20 part approaches.

Processing a large file without splitting it generally leads to slower performance due to limited memory and CPU resources. While dividing the file into 2 parts for parallel processing is faster than processing it as a whole, it remains the slowest option due to limited parallelism and process management overhead. Splitting the file into 5 parts improves performance, but it’s still slower than dividing it into 10 or 20 parts. Dividing the file into 10 parts strikes a good balance, reducing processing time significantly compared to 2 or 5 parts, while 20 parts yields similar results with marginal improvement. After a certain point, further splitting provides diminishing returns.

While splitting a file into multiple parts can enhance processing efficiency, additional strategies can further reduce processing times. One such approach is using parallel libraries like Dask, which enables the efficient handling of large datasets by breaking them into smaller chunks and integrates seamlessly with Python's multiprocessing capabilities. Another method involves asynchronous I/O operations through Python's asyncio library. Which allows for non-blocking file reading and writing so the program can execute other tasks while waiting for I/O to complete. For very large files, distributed computing via cloud platforms like AWS or Google Cloud Functions can further speed up processing by distributing the workload across multiple machines.

In conclusion, the experiment found that splitting the file into 10 or 20 parts and processing them in parallel resulted in the fastest processing times. These methods were significantly faster than breaking the file into 2 parts, which was the slowest method. While splitting the file into 5 parts did show some improvement, it was still slower than the 10 part or 20 part methods. To further improve processing times, strategies such as using parallel processing libraries(Dask), asynchronous I/O, and distributed computing could be implemented. These approaches would enable more efficient use of system resources and further reduce processing time for large datasets.



*Output with runtimes where n is the number of parts the input file is broken up into.*

### References

Dask Documentation. (n.d.). Retrieved from<https://docs.dask.org/>

Python Software Foundation. (2024). asyncio — Asynchronous I/O. Retrieved from<https://docs.python.org/3/library/asyncio.html>

Lee, J. (2021). Efficient parallel data processing using Python. *Journal of Data Science and Computing*, 12(3), 45-59.